Power Laws in Object Oriented Systems Architectures

Nicola Serra

2005

Contents

List of Figures

List of Tables

Acknowledgment

During my PhD course I have been working as a part of a wide group, whose main research topics have been and still are located in the field of Software Engineering, with a special interest and attention to the Agile Methodologies and to the software systems and processes modeling. My work, study and research activities have been always related to the work, study and research activities of other people within such group. Thus, I must and want to thank all of them. Especially, thanks to Prof. Michele Marchesi for his guidance in the course of my research and academic experience. Thanks to the great Stefano Chiodo Tuveri, who actively contributed to the design and implementation of the parsing tools and to the the analysis of data results. Thanks to the Agile Group and to the Agile Gruppo.

Grazie alla mia Famiglia, sempre presente.
Senza di loro non sarebbe stato possibile.

Introduction

Many real systems have been described as complex networks, where nodes represent specific parts of the system and connections represent relationships among them. Examples of such networks come from very different contexts. For example, technological networks such as the Internet [Faloutsos, 1999][Chen et al, 2002], the World Wide Web [Broder et al., 2000] and the North American Power grid [Amaral et al., 2000] have been analyzed as complex networks. Many other examples come from context such as social science, where network of acquaintance or other relationships between individuals have been modeled and analyzed as complex networks [De Castro and Grossman, 1999] [Newman, 2001]. Also biological systems such as neural and metabolic networks, blood vessels and protein interaction webs have been modeled following an analogue approach [Jeong et al., 2001].

The study of networks in the form of mathematical graph theory is a fundamental topic of the discrete mathematics and it has a long history, as it starts in the eighteenth century with Euler. More recently, an important contribution to the study of such systems came from social sciences, whose interest started to emerge already around 1930s. However, recent years have witnessed a substantial new movement in the study of complex networks. Driven by the the fast increasing of computational power, the focus has been in fact shifted from the study of a single relatively small graph to the analysis of the statistical properties of families of large systems, made of thousands of nodes and intricately connected by millions of connections.

Traditional theories suggest to represent complex systems as random graphs, according to the models proposed by Erdős and Rényi [Erdős and Rényi, 1959] [Erdős and Rényi, 1960]. However, there is an increasing evidence that many real world systems behave displaying global statistical properties that are not accounted by the random graph model. More precisely, despite the very different realms where complex systems can be found, almost all of them show a high degree of self-organization into a scale-free state [Barabasi and Albert, 1999] [Barabasi et al., 1999]. In practice, when these systems are modeled as graphs where the vertices represent the entities of the system, and the (possibly oriented) edges the relationships between them, the distribution of edges connected to vertices follows a power-law. Power laws, also known as Pareto distributions, are not a characteristic that needs a graph to be found, they are also found in the distribution of features related with single entities. The distribution of words in a corpus of documents, of individuals and firms wealth, of stock market daily returns, and even of the size of U.S. cities, all follow a power-law.

Inspired by the large interest of research on complex networks, recently it has emerged the interest in applying such theories and models to represent large software systems. Many software systems have reached such a huge dimension that it looks sensible to treat them as complex networks [Focardi et al., 2000]. Software is built up out of many interacting units and subsystems at many levels of granularity (functions, classes, interfaces, libraries, source files, packages, etc.), and the various kinds of interactions among those pieces can be used to define graphs that form a skeletal description of a system. Moreover, these entities are characterized by features whose distribution in turn can be studied looking for scale-free behavior.

Some authors have already found significant power laws in software systems. Potanin studied the graphs formed by runtime object oriented programs written in a variety of languages, namely Java, C++ and Smalltalk, and showed that these turn out to be scale-free networks without exception [Potanin et al., 2002]. Valverde studied the emergence of scaling in software architecture graphs belonging to large Java and C/C++ open source software systems [Valverde et al., 2002] [Valverde and Solé, 2003]. The graphs he studied did not take into account detailed relationship semantics, and did not make any difference between the various kinds of relationships among classes. He consistently found significant power laws in the graph vertices input and output edge distribution. Myers found the same properties, again studying open-source software systems [Myers, 2003]. Wheeldon and Counsell studied various kinds of class properties and relationships of large Java systems, finding many power-law distributions [Wheeldon and Counsell, 2003]. Marchesi studied and again found scale-free behavior within Smalltalk systems [Marchesi et al., 2003].

Besides the intrinsic interest of finding scaling laws in software systems networks, the study of their complex structure can also be interesting from the perspective of the software engineering. Complex network theory, while applied to represent software systems, provides in fact an alternative and new perspective for the analysis of many software systems properties. This could be achieved for example correlating the topological properties of the underlining network with various quality indicators or general size and complexity metrics computed on the actual represented software system. Nevertheless, while all of the previous works gave evidence that software system networks display connectivity with scale free distribution tails, in accordance with the statistical properties of many other real networks, the practical implications of such results on the development and evolution of software systems are poorly accounted.

Starting from the above observations, this work aims to a double objective. First, to study the general statistical properties of the network representation of a software system and, second, to provide an interpretation of such analysis within the boundaries of software engineering. More precisely, we will focus on the representation of the object oriented software systems and on the analysis of their topological graph properties in terms of software quality.

So far, the application of complex networks theory to the representation of software systems have been mainly focusing on evaluate whether or not the degree distributions of the software graphs follow a power law. This work goes further, providing evidence that such distributions can be also found for other software properties not requiring a graph representation. Moreover, it will be shown how the degree distribution themselves can be fitted also by other types of distribution such as lognormal and sometimes normal functions. Nevertheless, the best contribution to the state of the art is given by the correlation analysis of the network topological properties with the most important object oriented software quality indicators. This for the first time provides a quantitative and formal interpretation of the global statistical properties displayed by software networks and shows that the scale free behavior of software networks reflects the scale free behavior of well defined software properties, which are in turn independent of the graph representation.

Chapter One gives an overview of the software measurement theory and presents the main object oriented metrics proposed in literature, namely the Chidamber and Kemerer metric suite [Chidamber and Kemerer, 1994]. On the specific, it presents the main literature results relative with the empirical validation of such metrics in terms of code quality. This provides a basis for the interpretation of further results within the terms of a software engineering approach. More details about other software metrics adopted in this study are given in **Appendix A**. **Chapter Two** presents the main issues around the complex network theories. The main results will be presented, starting from the *small world effect* to the *emergence of scaling* properties revealed by real networks. A specific section is dedicated to give an overview of the application of the complex network theory to the representation of software systems. **Chapter Three** describes the approach followed to perform the statistical analysis and also provides a description of the method adopted for the definition and implementation of the software graph representation. The exaustive descriptive statistic of each analyzed system is provided in **Appendix B**. **Chapter Four** is dedicated to the presentation and discussion of the analysis results. Finally, **Chapter Five** provides the conclusions and proposes some cue for further works. The study has been performed both on Smalltalk and Java systems.

Both modeling and metric extraction required the design and implementation of specific parsing tools. Few details about tools architectures are given both in **Chapter Three** and in **Appendix A**. Nevertheless, the deep explanation of the parsing tools implementation is not explicitly accounted here, where I chose to give more relevance to the modeling aspects. The work presented here represents the main part of the research activity of my PhD course, in fact focused on the study of graph based models for the representation of object oriented software systems and on the implementation of parsing tools for the extraction and definition of software metrics.

Chapter 1

Measures in Software Engineering

Measurement is an inherent and fundamental activity of all engineering disciplines. Measurement provides the ability to control activities, products and resources of a specific process.

Compared with other traditional engineering disciplines, software engineering is for sure the youngest and most immature. On the other hand it is surprising how fast, above all during the last decade, software engineering has increased its power to deliver and produce large, reliable and high quality software systems. Always new methodologies, techniques, process models and tools have been proposed and successfully applied to improve and support the production of high quality software systems.

The aim of software engineering is to provide the technologies which apply an engineering approach to the development and support of software products and processes. Software engineering activities includes managing, planning, modeling, analyzing, designing, implementing, testing and maintaining. *Engineering approach* means to *well define in order to control* all these process activities. To achieve this goal measurement plays a key role, allowing to understand, control and improve software development processes and products.

Measurement provides the mechanism to create the aid in answering a variety of questions associated with the enactment of any software process. It helps to support project planning, to determine the strengths and weaknesses of the current processes and products, to evaluate the quality of specific processes and products. Measurement also helps, during the course of a project, to assess its progress, to take corrective action based on this assessment, and to evaluate the impact of such action.

Moreover, measurement becomes essential to provide visibility to entities and relationships, to quantitatively assessing problems and to suggest adequate solution scenarios. It happens at all different levels of details and abstraction, helping engineers to make the better decision about methodologies, techniques and tools to improve or to simply understand the status of processes, products and resources.

This section will give a brief introduction about the software measurement practice. The first part will provide the background theory of software measurement. Then the focus will be moved on the specific of the object oriented paradigm and the main object oriented metrics will be presented and discussed.

1.1 Software Metrics Classification

The first obligation of a measurement process is to establish what to measure. This is a general concept whose validity also address software engineering. Thus, the primary objective in applying a measurement approach upon software development is to identify the entities and attributes to be measured.

Following Fenton [Fenton, 1997] software entities may be of three different types:

- **Processes** are collections of related software engineering activities.
- **Products** are any artifacts produced as output of any software engineering activity.
- **Resources** are any artifacts required as input of any software engineering activity.

Typically, the definition of a process is related with a timescale. Thus, the activities of a process are related in some way which depends on time. Resources and products are related with processes. Each process activity is characterized by the resources it uses and by the products it produces. Any artifact can be considered both a resources or a product according to the specific associated process. More specifically, an artifact produced by a specific activity can feed another activity.

Measurement is the activity of quantify in some way any attribute of a process, product or resource. Given any specific activity or artifact to measure, we can distinguish two different types of attributes:

- **Internal Attributes** of a process, product or resource are those that can be measured in terms of the sole process, product or resource. Thus, internal attribute can be measured considering the process, product or resource on its own, independently of its behavior.

- **External attributes** of a process, product or resource are those that can not be measured in terms of the sole process, product or resource. Thus, external attribute can only be measured considering the process, product or resource related with the external environment and context. In this case the behavior becomes as important as the entity itself.

For any industrial process quality and productivity are indeed the most important parameters to be controlled and quantitatively assessed. This is true for industrial software development, too. Nevertheless, a formal definition of terms such like *quality* and *productivity* is far to be simple. Typically, business level roles, such as managers and customers, are more interested on external attributes, because quality and productivity are expressed in terms of such kind of attributes. For example, Boehem expressed software quality in terms of *portability*, *reliability*, *efficiency*, *human engineering*, *testability*, *understandability* and *modifiability*. Thus, quality attribute is decomposed in other external attributes, which in turn are still far to be simply definable. In practice, the decomposition goes on in order to finally express external attributes in terms of internal attributes, which are directly measurable. Similarly, McCall proposed an alternative model, which became the basis for the ISO 9126 software quality standard. The standard express quality again in terms of external attributes like *functionality*, *reliability*, *usability*, *efficiency*, *maintainability* and *portability*.

While external attributes are highly important on business level, internal attributes are fundamental as they are the ones that can be directly measured. Internal attributes represent the last step on the decomposition of external attributes, that is, external attributes are typically quantified in terms of internal attributes. Early and modern approaches such like top-down design, low-coupling-high-cohesion structured design, object orientation and design patterns are all based on a common principle: *good internal structure leads to good external quality*. This principle underlines the tight relationships between internal and external attributes and points that the establishment of a good level of desirable internal attribute leads to the consequent establishment of a good level of the high abstraction desirable attributes.

1.2 Objectives of Software Measurements

There are compelling reasons to consider measurement as a fundamental activity of software production process.

Following Fenton [Fenton, 1997], independently of the specific perspective, measurement is fundamental to achieve three main objectives:

- **Understanding**
- **Control**
- **Improvement**

First of all, measurement allows to asses the current situation about processes or products within a given development phase or within a more specific context. In this sense, measurement makes aspects of process and products more visible, providing a better *understanding* of entities, relationships and activities. This helps the involved roles to establish the baselines to set the goals for future behaviors.

Consequently, measurement becomes the basis to *control* the software development processes and products. Starting from the assessment of the current status and then establishing the future goals, it becomes possible to make predictions about what is likely to happen and to perform changes and corrections to processes, products and resources, in order to maintain the evolution inside acceptable bounds.

Last, measurement supports and encourages *improvement* of processes and products. For example, the quantitative assessment of the current product quality level, or process productivity is the starting point to search for new solutions in order to increase the same levels.

The early establishment and definition of the objectives in a measurement process is the basis of the *Goal Question Metric* (GQM) measurement framework [Basili and Rombach., 1988] [Basili, 1992]. According to the GQM framework, measurement is a top-down process, which start defining a set of operational goals as a basis to select the appropriate set of metrics to achieve those goals. There are a variety of uses for measurement, thus the purpose should be clearly stated. Moreover, measurement needs to be viewed from the appropriate perspective. The manger, the developer, the customer and the final user each view the final product in a different perspective, thus they may want to know different things about the project and with a very different level of detail. The measurement model is based on a set of tree different levels:

- **Conceptual level - GOAL:** a goal is defined for an object from a specific point of view, relative to a particular environment. Objects of measurement may be products, processes or resources.
- **Operational level - QUESTION:** a set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model. Questions try

to characterize the object of measurement (product, process, resource) with respect to a selected quality issue and to determine its quality from the selected viewpoint.

- **Quantitative level - METRIC:** a set of data is associated with every question in order to answer it in a quantitative way

The process of setting goals and refining them into quantifiable questions is critical and complex and requires a great level of experience. In order to support such step, Basili provided both the templates for the definition of the conceptual level and the guidelines for deriving questions and metrics [Basili, 1992].

1.3 Object Oriented Metrics

Management of complexity plays a key role in the development of high quality software systems. While it is nowadays well accepted that there is *no silver bullet* to deliver the perfect system [Brooks, 1987], also there is the evidence that the Object Oriented paradigm has been playing a fundamental role in software engineering, as an optimal technology to successfully apply the *divide et impera* principle for managing high complexity of large systems.

According to the object oriented paradigm a software system is made of *objects* which interact one to each other by passing *messages*. A software object is the abstraction of a real object within an existing real domain. Each object is characterized by its *state* and *behavior*, the first describing the internal information structure and the second the way it can interact with external environment.

An object oriented system starts by specifying a *class* containing the definition of related *attributes* and *operations*, that is the definition respectively of the state and behavior of its instances (objects). Thus, a class is used as the basis to instantiate objects. In this sense, a class plays the role of a common interface shared by all its objects.

Classes and objects can be related across different mechanisms and relationships. First, classes and objects are related across *inheritance* relationship. A child class inherits all of the attributes and operations from its ancestor classes, having in addition its own attributes and operations. A child class can also become a parent class for other classes, forming another branch in the hierarchical tree. Second, objects interact by passing *messages*. When a message is passed between two objects, the object classes are said to be coupled.

The application of object orientation pervades all software development process activities, from the requirement specification, to the analysis and design, to the implementation. Obviously, object orientation itself cannot be considered the warranty of good software delivery, if not supported by an adequate engineering approach.

The object oriented paradigm inherits part of its concepts, such like coupling, cohesion and information hiding, from traditional modular software engineering approaches [Parnas, 1972]. On the other hand, there are concepts, such like inheritance and overloading, which are new and object oriented specific. While other general software concepts, such like size and complexity, need to be reinterpreted in the perspective of object oriented approach. Thus, a formal definition of object oriented structure attributes and concepts is needed, in order to achieve the application and the extension of software measurement principles to object oriented systems. Moreover, object oriented approach requires the definition and validation of ad-hoc metrics, which should be able to measure processes and product attributes of object oriented software systems.

This section will present and discuss the main object oriented metrics proposed in the software engineering literature. Metrics will be presented and compared on the basis of their practical implementation and theoretical validation.

1.3.1 CK Metrics Suite

Chidamber and Kemerer [Chidamber and Kemerer, 1994] have suggested a suite of six object oriented design metrics. The theoretical basis for the definition of such metrics, is the set of *ontological principles* proposed by Bunge [Bunge, 1979] and later applied to the formal definition of object oriented systems by Wand and Weber [Wand and Weber, 1990]. The Bunge ontology has largely interested researchers involved in the formal modeling of object orientation, since it deals with concepts inherent to the real world, which is the claimed basis of object oriented approach. According to Bunge ontology the world is viewed as composed of *substantial individuals* having a finite set of *properties*. These concepts can be formally translated and encapsulated in the object oriented paradigm definition, where an object is equivalent to a substantial individuals collectively considered with its properties. Similarly, attributes such as coupling, cohesion, complexity and inheritance can be defined in terms of Bunge ontology. This approach provides for the first time a theoretical and formal basis of metrics definition. As pointed by Chidamber and Kemerer themselves, not only previous attempt to define object oriented metrics, but more generally the large part of traditional metrics definition lacked such theoretical basis.

In this section we will present the CK Suite metrics and discuss them in terms of their implementation and empirical validation. All the following definitions are directly referred to the original paper.

- **WMC - Weighted Methods per Class**

  Definition: consider a class C, with methods M_1,...,M_n that are defined in the class. Let c_1,...,c_n be the complexity[1] of methods. Then:

  $$WMC = \sum_{i=1}^{n} c_i \tag{1.1}$$

  If all method complexities are considered to be unity, then WMC is the number of methods (NOM) for the class C:

  $$WMC = NOM = n \tag{1.2}$$

  Interpretation: this metric provides a measure of the complexity of a class. Complexity is intended in terms of Bunge ontology, where it is defined as the cardinality of properties of a substantial individual. The number of methods and their complexity are predictors of the effort required to develop and maintain the class. The larger the number of methods, the greater the potential impact on children classes, as they inherit all methods of parent classes. Moreover, a class with a large number of methods are likely to be application specific, limiting the possibility of reuse.

- **DIT - Depth of Inheritance Tree**

  Definition: the DIT metric is the number of ancestors of a given class, that is the number of nodes (classes) to be crossed to reach the root of the inheritance tree. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.

  Interpretation: DIT is a meusere of how many ancestors may potentially affect a class. The deeper a class is in the hierarchy tree, the greater the probability of inherit a large number of methods, thus increasing class complexity. On the other hand, the deeper a class is in

[1]Complexity is deliberately not defined more specifically here in order to allow for the most general application of this metric.

the inheritance tree, the greater the potential level of reuse of inherited methods.

- **NOC - Number of Children**

  Definition: the NOC metric is the number of immediate subclasses subordinated to a class in the class hierarchy.

  Interpretation: NOC is a measure of how a class can potentially affect its subclasses. The greater the number of children, the larger the level of reuse. On the other hand, a high value for this metric may warn a misuse of inheritance mechanism.

- **CBO - Coupling Between Object Classes**

  Definition: the CBO metric for a given class is a count of the number of other classes to which it is coupled.

  Interpretation: two classes are coupled when methods declared in one class uses methods or instance variables defined by other class. High coupling negatively affects modularity of the design. In order to promote encapsulation and improve modularity, class coupling must kept to the minimum.[2]

- **RFC - Response For a Class**

  Definition: the RFC metric is the cardinality of the response set for a class. Given a class C, with methods M_1,...,M_n, let R_i with $i = 1, .., n$ to be the set of methods called by M_i. Thus, the *response set* for the class C is defined as:

  $$RC = \bigcup_{i=1}^{n} R_i \tag{1.3}$$

  the RFC metric is then given by:

  $$RFC = |RC| \tag{1.4}$$

  Interpretation: the response set for a class is the number of methods that can be potentially executed in response of a message received by

[2]Low coupling as indicator of good design does not belong only to object orientation approach. Rather, it comes from the modular design theory, formally proposed by Parnas[Parnas, 1972].

an object of that class. This is a measure of the communication of the class with other classes in the system. The greater the number of methods that can be invoked by a class, the larger the complexity of the class. RFC gives a measure of the effort required for maintain a class in terms of testing time.

- **LCOM - Lack of Cohesion in Methods**

  Definition: Given a class C, with methods M_1,...,M_n, let I_i with $i = 1,..,n$, to be the set of instance variables accessed by M_i. We define the *cohesive method set* and the *non-cohesive method set* as:

$$CM = \{(M_i, M_j)|I_i \cap I_j \neq \varnothing, i \neq j\} \tag{1.5}$$

$$NCM = \{(M_i, M_j)|I_i \cap I_j = \varnothing, i \neq j\} \tag{1.6}$$

  the LCOM metric is then given by:

$$LCOM = \begin{cases} |NCM| - |CM| & \text{if } |NCM| > |CM| \\ \\ 0 & \text{otherwise} \end{cases} \tag{1.7}$$

  Interpretation: the LCOM metric is refers to the notion of *degree of similarity* in methods. The degree of similarity is formally given by:

$$\sigma(M_i, M_j) = I_i \cap I_j \tag{1.8}$$

  The larger the number of similar methods, the more cohesive the class. This approach is coherent with traditional notion of cohesiveness, which is intended to account the inter-relatedness of different part in a program. Cohesiveness of methods in a class is a desirable attribute, since it promotes encapsulation. Lack of cohesion implies classes should probably split into two or more specialized subclasses.

1.3.2 CK Metrics as Quality Indicators

The object oriented approach has been claimed as a key technology to better manage software complexity and to provide improvement of systems quality. Software engineering researchers spent great efforts to empirically validate such claims. The object oriented research mainly has been studying the relationship between object oriented metrics and quality in terms of defects, specifically those reported during customer acceptance testing. Defect density is considered to be an indicator of fault proneness at class level and then a measure of maintenance effort. Similarly, defect density is reasonably related with extent of code change, which is again a surrogate measure of maintainability.

Early studies found the existence of a relationship between defects and both traditional complexity measures, such as the McCabe ciclomatic complexity, and size measures, such as the LOC family metrics. More recently many empirical experiments have been performed to find similar relationships between defects and object oriented metrics such like the Chidamber and Kemerer metrics previously described.

Li and Henry investigated the six metrics, including some simple size metrics, on two commercial projects to determine whether such metrics can be linked to the extent of code change, which in turn has been considered as an indicator of maintenance effort. On the basis of an empirical study based on regression analysis, they concluded these metrics to be indeed useful for such purpose [Li and Henry, 1993]. Following Li and Henry, Basili and his colleagues investigated the CK suite on eight medium-size projects performed in an academic context. On the basis of a statistical regression analysis, they gave evidence on the ability of such metrics to be early quality indicators in terms of fault-proneness prediction at class level [Basili et al., 1996]. On three financial applications, Chidamber found that coupling and cohesion metrics are in fact related with productivity, rework and design effort [Chidamber at al, 1994]. Briand and his colleagues, investigated all CK metrics and again they found a relationship between all metrics of the suite, except for LCOM, and fault proneness at class level [Briand et al., 1999][Briand et al., 2000]. Similar results have been confirmed by Subramanyam and Krishnan on data collected from an industrial context [Subramanyam and Krishnan, 2003]. More recently, Alshayeb and Li performed an empirical study validating DIT, WMC, LCOM and other object oriented metrics as predictors of evolution changes in short iteration processes [Alshayeb and Li, 2003]. They found such metrics to be good predictors of maintenance effort in Extreme Programming environments.

While there is large number of studies which validate CK metrics on empirical basis, also literature presents some critics mainly on theoretical basis. Churcher and Shepperd criticized CK metrics on the basis that there is no consensus on the definition of the underlining measured attributes [Churcher and Shepperd, 1995]. They argued that even the notion of *method per class* is ambiguous, in relation of the specific adopted language. Nevertheless, Chidamber and Kemerer themselves pointed that this lack of precision in the definition of the attributes is due to the fact that all metrics of the suite are intended to be design metrics. This implies the metric definition to be independent of the specific language implementation. Henderson-Sellers and his colleagues [Henderson-Sellers et al., 1996] criticized the definition of CBO and LCOM as lacking of mathematical basis and proposed their alternative definition. Moreover, CK metrics did not account for complexity which can arise from object oriented design factors such like polymorphism and encapsulation. An example of metrics suite accounting such factors is the MOOD suite proposed by Abreu [Abreu and Carapuca, 1994][Abreu et al., 1995].

More metrics, with some implementation detail, will be provided in Appendix A.

Chapter 2

Complex Networks

2.1 Introduction

A *network* is made of a set of entities, which are usually called *vertices* or *nodes*, and a set of connections between them, which are usually called *edges*. There is a great number of real systems that can be and in fact have been described as complex networks. Examples include the Internet, the World Wide Web, social networks of acquaintance between individuals, organizational networks and networks of business relations between companies, metabolic networks, food webs, distribution networks such as the postal delivery routes, networks of citations between papers, and many others.

The origin of networks theory starts in the far 1735, when Euler gave the solution of the Kőnigsberg bridge problem. This is often cited as the first proof in the theory of networks and as the seed of the mathematical graph theory. The mathematical graph theory has been largely developed during the twentieth century and networks have also been largely studied in the social sciences. Although the graph theory and the interest in networks have a remote origin, during recent years has emerged a new approach in the study of such systems. What makes the recent approaches new, if compared with the previous traditional studies, is the large extension of the considered networks. This shifted the focus from the properties of a single small graph, or even of a single node, to the global statistical properties of the entire network. Moving from small graphs to billion nodes networks represents the boundary between the traditional graph theory and the modern complex networks theory. Moreover, this is the main characteristic which makes a simple network to become complex.

The body of the new approach aims to a double goal. First, to understand the global statistical properties of complex network, in terms of the distributions of topological entities such as the nodes degree. Second, to extract network models in order to understand the meaning of such properties and how they emerge.

The very first attempt to model complex networks in terms of statistical properties is represented by the *random graphs* theory proposed by the two Hungarian mathematician Erdős and Rényi [Erdős and Rényi, 1959], [Erdős and Rényi, 1960].

Using the classical random graph theory to model complex networks looks sensible, but in fact a great number of real systems behave according to laws significantly different from those predicted by Erdős and Rényi theory.

More precisely, there is increasing evidence that several real networks behave as *small worlds*, simultaneously showing short length path and high clustering behaviors [Watts and Strogatz, 1998]. 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 [Barabasi and Albert, 1999].

Large software applications are considered to be among the most complex artifacts ever produced by man [Brooks, 1995], and consequently are good candidates to be modeled as complex networks. This is particularly true for object-oriented systems, where objects and classes are natural candidates to be represented as nodes, while the various possible relationships between them – such as inheritance, instantiation, composition, dependence – can be represented as arcs connecting nodes.

Very recently, some studies have been performed on software systems showing that run-time objects [Potanin et al., 2002] and static class structures of object oriented systems are in fact governed by scale free power law distributions [Valverde et al., 2002][Valverde and Solé, 2003]. Other studies have been performed by Myers [Myers, 2003] and Wheeldon and Counsell [Wheeldon and Counsell, 2003], leading to similar results. While, most of these studies of complex software systems are based on static languages such like C++ and Java, Marchesi [Marchesi et al., 2003] provides evidence that similar behaviors are displayed also by dynamic languages such like Smalltalk.

2.2 Networks Models and Properties

This section will present the main differences between early and modern networks theories, starting from the random graph model and then moving to small world and scale free models. Some real system examples will be provided.

2.2.1 Random Graphs

The random graphs theory have been first proposed by Solomonoff and Rapport [Solomonoff and Rapport, 1951] and independently around 60's by the two Hungarian mathematicians Erdős and Rényi [Erdős and Rényi, 1959] [Erdős and Rényi, 1960].

Following Erdős and Rényi, a random graph $G_{n,p}$ is simply made staring from a given set of n nodes, then considering each possible pair and independently drawing a connection with probability p or doing nothing with probability $1-p$. Formally, $G_{n,p}$ is the family of all graphs where a single instance with m connections appears with probability $p^m(1-p)^{M-m}$, where $M = \frac{1}{2}n(n-1)$ is the maximum possible number of connections. Erdős and Rényi proposed the alternative model $G_{n,m}$, which is the family of all the graphs having n nodes and exactly m connections, where each graph instance appears with equal probability. The two models are equivalent and quite all properties displayed by one can be translated to the second, if preserving the condition of independence and in the limit of large number of nodes n.

In a series of articles, Erdős and Rényi gave proof of many properties of random graphs . All properties are exactly solvable in the limit of large graph size [Erdős and Rényi, 1959][Erdős and Rényi, 1960][Erdős and Rényi, 1961].

One of the main characteristic of a random graph is the Poisson distribution of the node degree. The degree of a node is the number k of edges directly connected with the node itself. Since the presence or absence of a connection between each pair of nodes is independent, the probability of a node to have k directly connected edges in a graph of n nodes is given by:

$$p_k = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} \cong \frac{z^k \cdot e^{-z}}{k!} \tag{2.1}$$

The second approximation becomes exact in the limit of large n and holding the mean degree $z = p(1-n)$ constant. The degree distribution given in equation 2.1 is the reason why the above random graph model is also called *Poisson Random Graph*.

The expected behavior of a random graph varies with the value of p. Erdős and Rényi demonstrated what is considered the most important property of a random graph, that is the existence of a transition phase from a low-p state where there are lots of small disjoined elements with a few number of connection, to a high-p state where the large part of nodes are joined together in a single *giant element*. The transition phase can be easily explained following Janson, who provides an alternative evolution model of a random graph [Janson, 2000]. Following Janson, a random graph can be easily represented as a growing process where, starting from a given set of n nodes, edges are added one by one. Each connection is chosen uniformly at random among all possible remaining pair of nodes within the graph. According to this model and adding each edge at fixed times $t = 1, 2, 3, ...$, we obtain the $G_{n,m}$ model exactly at time m, while adding each edge at random time uniformly and independently taken from the interval $(0, 1)$, we obtain the $G_{n,p}$ at the time $t = p$. The evolution of the graph starts from the n nodes without any connection. During the entire process the graph becomes first a *forest of small trees*, then the *giant element* appears and at the end the process lead to the birth of the *complete graph*, that is the graph with all the possible connections. The study of the behavior near the transition phase was treated first by Solomonoff and Rapport and then by Erdős and Rényi. More recently others such like Luczak [Luczak, 1990][Luczak et al., 1994][Luczak, 1996] and Janson [Janson, 2000] deeply explained this issue.

For an exhaustive mathematical treatment of random graph theory see the books by Bollobás [Bollobás, 1995] and by Janson, Luczak and Rucinski [Janson et al., 1999].

2.2.2 Small World Networks

Given an undirected network, we define the *mean geodesic length* between vertex pairs as follow:

$$l = \frac{2}{n(n+1)} \sum_{i \geq j} d_{ij} \qquad (2.2)$$

where d_{ij} is the geodesic distance between vertex i and vertex j, that is the minimum number of adjacent edges that links vertex i and vertex j. The geodesic distance between two vertexes is also known as *short length path*. Consequently, the mean geodesic length given in equation 2.2 is also called *mean short length path* of the network. The definition given in equation 2.2 becomes problematic while applied on networks having disjoined elements. In this case, two nodes which respectively belong to disjoined subgraphs have infinite geodesic distance, that is the mean geodesic distance l becomes also infinite. To avoid this problem, pairs that falls in disjoined subgraphs are typically excluded from the average.

A large number of real networks display a small value of the mean geodesic length l, if compared with the number of nodes n. For examples, many social networks such as the movie actors network, where Hollywood actors are linked if they have worked together for the same movie, the science coauthorship networks, where scientists are liked if they are coauthors of the same paper, display values of l between 3.5 and 7.5. Also, many information, technological and biological networks display similar values (Table 2.1).

The small value of the mean geodesic length is known as *small world effect*. The small world effect was described for the first time in the 60's with a famous experiment carried out by Stanley Milgram [Milgram, 1967]. The experiment consisted in passing a letter from person to person to reach a designed target, with the constraint that one individual had to pass the letter only to a first-name acquaintance. The experiment showed that a any generic target could be reached staring from any far individual with around six steps. This result is known as the *six degree of separation*.

The small-world effect has a great impact and implication on real networks. Mainly, it plays a fundamental role on the dynamics of network, since it impacts the fastness of spread across the network itself. For examples, the rapid spread of information around the Internet, the efficiency of a postal delivery system or the time it takes for a disease to spread across a population are all closely related to existence of the small world effect on the underlining network.

Recently, the term small world effect has assumed a more formal meaning. Mathematically, we say that a network shows the small world effect if the value l scales logarithmically or slower with the network size n and for fixed mean degree. Logarithmic scaling can be proved for a variety of network models. For examples, Bollobás [Bollobás, 1981] and Chung and Lu [Chung and Lu, 2002] showed that random graphs in fact display such behavior, while Bollobás and Riordan [Bollobás and Riordan, 2002] provide proof that networks with power law degree distributions behave displaying the small world effect.

While random graph model reflects the small world effect displayed by many real networks, a clear deviation of random graphs is the *transitivity* of actual nets. Transitivity, also known as *clustering*, reflects the presence of a high number of triangles in the network, where a triangle is a set of three vertices connected one to each other. That is, if the node A is connected to the node B and the node B is connected to the node C, then there is an high probability for the presence of a link between the node C and the node A. In social terms, this means that *the friend of your friend is also likely to be your friend*.

Mathematically, we define the *local clustering coefficient* for a given node as:

$$C_i = \frac{NumberofActualLinksAmongNeighbors}{NumberofPossibleLinksAmongNeighbors} \tag{2.3}$$

If the node has degree k, the local clustering coefficient is a fraction of

Network	n	m	z	l	C	α	*refs*
Movie Actors	449,913	25,516,482	113.43	3.48	0.78	2.3	[Watts and Strogatz, 1998]
Math Coauthorship	253,339	496,489	3.92	7.57	0.34	-	[De Castro and Grossman, 1999]
Phys. Coauthorship	52,909	245,300	9.27	6.19	0.56	-	[Newman, 2001]
Biol. Coauthorship	1,520,251	11,803,064	15.53	4.92	0.60	-	[Newman, 2001]
WWW Altavista	203,549,046	2,130,000,000	10.46	16.18	-	2.1/2.7	[Broder et al., 2000]
Paper Citations	783,339	6,716,198	8.57	-	-	3.0/ -	[Render, 1998]
Internet	10,697	31,992	5.98	3.31	0.39	2.5	[Faloutsos, 1999][Chen et al, 2002]
Power Grid	4,941	6,594	2.67	18.99	0.080	-	[Amaral et al., 2000]
Protein Interaction	2,115	2,240	2.12	6.80	0.071	2.4	[Jeong et al., 2001]

Table 2.1: *Statistics for some real networks. Data are about the total number of nodes n, the total number of connections m, the mean degree z, the mean geodesic length l, the clustering coefficient C and the exponent α of the power law degree distribution for scale-free networks. A double value of the power law coefficient means a directed network, where both input and output degree are defined. The networks showed come from very different contexts. The movie actors network is a social network where each node is the representation of a movie actor. Two actors are linked if they have been working together on some movie. Math, physics and biological coauthorship networks are all social network, where scientists are linked together if they are coauthors on the same publication. The Altavista is the network representing the Altavista search engine database, where each node is the representation of a specific document on the web and a link is the representation of an hyperlink connection between two of such documents. The paper citation is an information network where each nodes represents a specific paper cataloged by the Institute for the Scientific Information between 1981 and 1997. Two paper are linked if one has been cited by the other. Follow the network representation of the Internet at level of autonomous systems and the network representing the power grid of the western north America. The last one is a biological network representing the interaction of proteins in the metabolism of the yeast S. Cerevisiae.*

$\frac{1}{2}n(n-1)$ in the interval $0 \leq C \leq 1$.

We also define a network global *clustering coefficient*, as the average of the local clustering coefficient computed for each node of the network:

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i \tag{2.4}$$

For the large part of real networks the clustering coefficient is considerably higher than for a random graph with the same number of nodes. This is one of the main limit that makes a random graph not suitable to model many real-world networks.

Watts and Strogatz overcame this limit introducing a new model of network still characterized by a small minimum length path, but simultaneously displaying a large clustering coefficient. They called their model *small world network* and proposed an interesting method to build such networks starting by a lattice [Watts and Strogatz, 1998]. Following their approach, we can start from a regular ring of L nodes each one connected with its k neighbors (Figure 2.1 - a). Then, we consider each edge and with probability p we rewire it moving one of its end to a new location chosen uniformly at random among the lattice nodes. This *rewiring* procedure perform a transformation of the well determined lattice structure, to a new one that is somehow random (Figure 2.1 - b). For $p = 1$ the lattice becomes something very close, while not identical, to a random graph. Figure 2.2 shows the variation of both minimum length path and clustering coefficient at the variation of p, that is, moving from a well defined structure such like the lattice to a random graph.

Watts and Strogatz showed how the random graph model is not suitable to represent certain real networks behaviors. Although their model is able to capture both clustering and small world effect, something more important is not still accounted. That is, the deepest gap between real networks and the early model proposed by Erdős and Rényi is about the shape of the degree distribution. While the theoretical model suggests the Poisson trend given in equation 2.1, many real networks in fact behave displaying heavy tail power law degree distributions.

2.2.3 Scale Free Networks

The term *scale free* refers to a functional form $f(x)$ which remains unchanged in response of the rescaling of the independent variable x. This implies a power law shape for $f(x)$, as this is the only solution of the equation

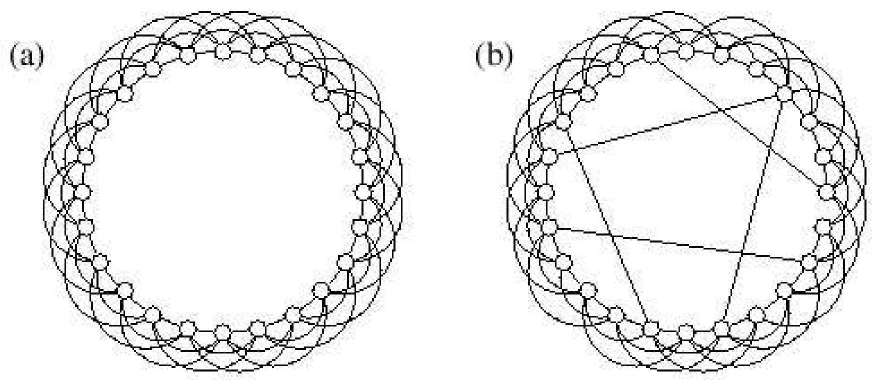

Figure 2.1: *Watts and Strogatz rewiring approach. Starting from a lattice (a), we rewire each edge moving one of its end to a new location chosen uniformly at random among the lattice node (b).*

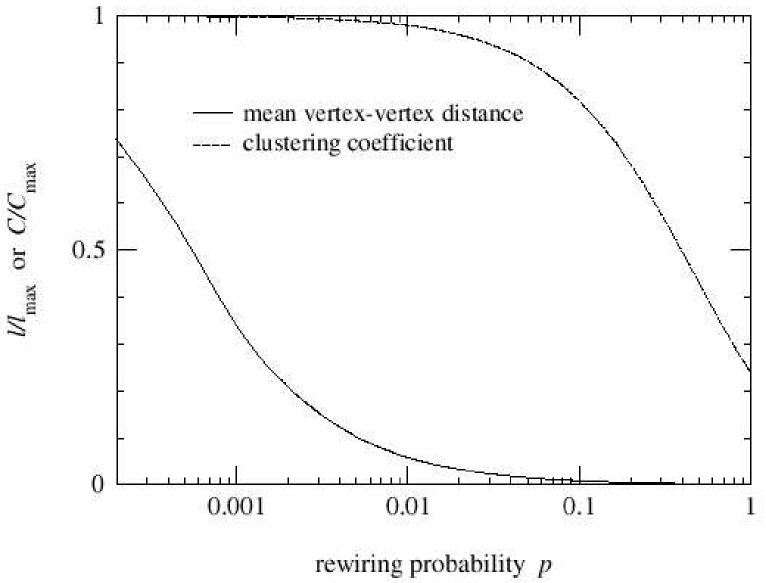

Figure 2.2: *The clustering coefficient and the minimum length path in the small world network model proposed by Watts and Strogatz. Between the two extremes, the $p = 0$ well determined structure and the $p = 1$ completely random structure, there is a region where the clustering coefficient is high and the minimum length path is small. This is the region where the small world rises up.*

$f(ax) = bf(x)$. Thus, the terms scale free and power law can be used for the same purpose.

Referring to networks, the term *scale free network* implies the considered network to have a power law degree distribution. That is, the probability for a node to have k directly connected edges is given by:

$$p_k \propto k^{-\alpha} \tag{2.5}$$

Power law degree distribution have been the focus of a great deal of attention. Recently, scale free behavior has been observed in a large number of real networks, including citation networks [Render, 1998], portions of the world wide web [Albert et al., 1999][Barabasi et al., 1999], the Internet [Faloutsos, 1999][Chen et al, 2002], telephone call graphs [Aiello et al., 2000], human sexual contacts [Liljeros et al., 1990], metabolism protein interaction networks [Jeong et al., 2001] and others. While some of these networks, such the WWW and the Internet, display a global power law shape within the whole degree distribution, many real networks show the scale-free trend effects only in the tail of the distribution. Figure 2.3 shows the degree distribution of some real networks.

According to the random graph theory the number of edges directly connected to a node is randomly distributed around an average value. The large part of the nodes have a number of edges close to this average, while the number of nodes having a number of connection substantially different from the average value decreases rapidly following an exponential trend. Real scale-free networks diverge from theoretical random model. The power law shape implies the network to have no average value of the node degree. Rather, there is a large number of nodes having a small number of connections, while there are few nodes very highly connected. A node with a large number of connections is called *hub*.

Barabasi and Albert [Barabasi and Albert, 1999] showed how the incongruence between the random graphs theory and real networks is due to two main factors not accounted for by the Erdős and Rényi model:

- Real networks expand continuously by the addition of new vertexes in contrast with the random graph model, which assumes a fixed starting number of nodes that remains unaltered while randomly adding connections between them.

- According to Erdős and Rényi each connection is added independently with the same value of probability, while many real networks grow up according to a preferential connectivity. More precisely, the probability for a new vertex to be connected to any existing vertex is not uniform. Rather, there is a higher probability that it will be linked to a vertex that already has a large number of connections, that is a hub.

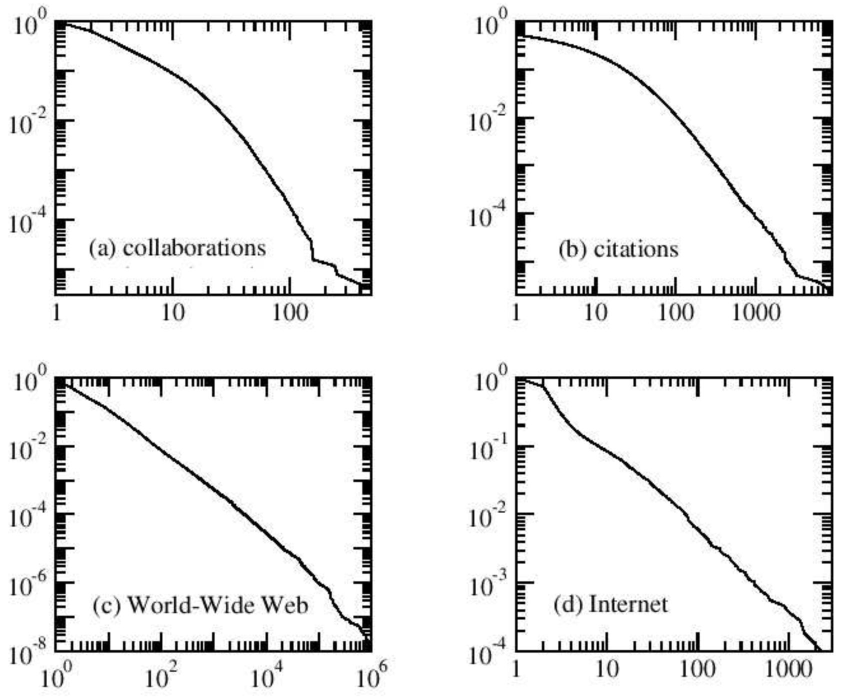

Figure 2.3: *Degree distributions of four real networks. Plots are displayed in logarithmic scale. This provides a better visualization, as logarithm performs a transformation of the power law into a straight line. Plots are respectively relative to the following networks: (a) mathematic collaboration network, (b) scientific paper citation network, (c) a subset of the World Wide Web, (d) the Internet considered at the level of autonomous systems. Two of these networks, (c) and (d), display a straight line in the log-log scale, which suggest a power law degree distribution across the whole interval. Two of them, namely (a) and (b), show a significant deviation from a power law in the small value region, while they still suggest a scale free behavior in the tail of the distribution.*

Barabasi and Albert gave proof that the above conditions are sufficient for displaying a power law tail in the distribution of the node degrees. Moreover, this *preferential attachment* model suggests a theoretical value for the power coefficient of $\alpha = 2.9 \pm 0.1$, which is quite close to the actual values empirically found for many real networks (see Table 2.1).

2.3 Software Systems as Complex Networks

The management and control of complexity is a main issue in the development of large software systems. In fact, complexity control is the first step to lead to high quality software systems. Software engineering has spent a great deal of effort to achieve this goal, always providing new technologies, methodologies and tools. *Decomposition* is historically a key factor in software complexity handling and it has been the main objective both of early development approaches such like the *modular decomposition* and the *abstract data typing*, as well as of the modern *object oriented* (OO) approach. Thus, typically software systems are build up out of many interacting modules and subsystems at many detail levels. For example, according to the object oriented paradigm a software system is made of objects which send messages one to each other, where an object is the abstraction of a real entity living in the represented domain. Similarly, following the more traditional structured programming, a software system is made of modules which interact one to each other across routines calls. Both object orientation and structured programming aim to build independent entities that perform all desired computation across different kind of collaborations and relationships. This modular structure, where software entities interact reciprocally suggest a graph based representation, where software entities can be represented as nodes and all different relationships, such like object interactions and routine calls, can be represented as connections between them.

Moreover, software applications and architectures have become ever larger and more complex over the past years. Thus, the idea of applying complexity and graph theories to model large software applications and to interpret their global statistic properties seems sensible.

Although the study of software systems as complex networks has a great scientific interest, to date it has received relatively little attention.

There is a main reason which awards a special role to software networks in the context of random networks theory. That is, software systems are explicitly build to be simultaneously highly functional and highly evolvable. Evolvability and functionality do not rise up as a peculiarity of the underlining network, rather are directly implemented following specific design practices such as responsibility-collaboration design and design patterns, or applying general optimization principles. Traditional principles, such as encapsulation and information hiding, low coupling and high cohesion design, and the modern object oriented approach are focused on the purpose of building high quality software systems, where quality is often pointed up in terms of functionality and evolvability.

This may suggest alternative scenarios for generating complex networks, which are more related to optimization rather than to preferential attachment or random generation hypothesis[Valverde et al., 2002].

On the other side, the analysis of software systems structure and evolution as complex networks could also be of practical interest from the software engineering point of view, as it might provide an alternative perspective able to help a better understanding of the mechanisms ruling software production and evolution, or the relationships among network structure and software quality. For instance, an entire new bunch of topological metrics computed on the network model could be introduced, and correlated with external software metrics.

Recently, few studies have been performed to explore how complex network theories applies to the modeling and representation of software systems, revealing that software networks display both small world and scale free effects. In one of the first studies of this kind, Potanin has shown 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 [Potanin et al., 2002]. Valverde found similar properties studying the graph formed by the classes and their relationships in large object-oriented projects. He found that software systems are highly heterogeneous small worlds networks with scale-free distributions of connections degree [Valverde et al., 2002][Valverde and Solé, 2003]. Wheeldon and Counsell performed similar studies on Java projects [Wheeldon and Counsell, 2003]. Myers found analogue results on large C and C++ open source systems, considering the collaborative diagrams of the modules within procedural projects and of the classes within the OO projects. He also computed the correlation between some size software metrics with graph topological measures, reveling that nodes with large output degree tend to evolve more rapidly than nodes with large input degree [Myers, 2003]. While, most of these studies of complex software systems are based on static languages such like C++ and Java, Marchesi provides evidence that similar behaviors are displayed also by dynamic languages such like Smalltalk [Marchesi et al., 2003].

There are still a lot of open issue. While all of the previous works gave evidence that in fact software system networks behave as small worlds and display connectivity with scale free distribution tails, the practical implications of such results on the development and evolution of software systems are poorly accounted.

Chapter 3

Experiment Overview

The goal of this study is to provide the analysis of an object oriented software systems under the light of the complex network theory. The work aims to a double objective, first to analyze the general statistical properties of an object oriented system in terms of the degree distribution of the underlining network, second to interpret the network topological informations in terms of software quality. Previous studies on this field mainly focused their attention on the analysis of the degree distributions, which is in fact the main topic of the complex network theory. Nevertheless, such studies lacked of practical and quantitative interpretation of the results, for example, in terms of statistical correlation between any measures directly performed on the topology of the network and any typical software metric usually computed to gather informations about the quality of the design.

Our study is based on the analysis of some object oriented systems developed using two different programming languages, namely Smalltalk and Java. This section will describe the various steps followed across this work. First, we will provide a description of the used approach to build the software graph. More precisely, we will separately consider the two cases of Smalltalk and Java systems, as the different nature of the respective languages ask for a different parsing code approach and for a different criteria in the mapping of the software entities upon the network elements. Second, we will describe and discuss the approach we used to analyze the degree distributions and how we have tested the scale free condition on them. Last, we will give some detail about the software metrics that we have correlated with topological graph properties and about the statistical approach we have adopted in the correlation analysis.

3.1 Building the OO Graph

According to the object oriented paradigm, a software system is the abstraction of a real domain, where objects interact reciprocally sending messages one to each other. The common interface of a family of similar objects is embodied in the class construct. Different classes can be related across different type of relationships such as inheritance, composition, aggregation, association and dependence. Not only classes and objects, but also many other elements, such as attributes, methods and interfaces, are defined in an object oriented system, representing entities at a different level of granularity and abstraction.

Thus, for any software system built according to the object oriented approach, it is possible to easily define different kind of networks made of nodes representing specific software entities and connections representing specific relationships between them. For example, it is possible to define the network made of the run time objects, where two class instances are connected as one send a message to the other; a static class network where a node is the representation of a class and a connection is the representation of a relationship such like inheritance or dependence; a more detailed network, where different types of node represent different types of entities, such classes, attributes and methods, and different types of connection are defined to represent specific relationships among them, such as instantiation between classes and attributes, calling between methods and so on.

This study will exclusively focus on the representation of classes and their relationships. Specifically, we will consider inheritance and dependence relations. We will describe the definition of the graph representation, respectively for Java and Smalltalk systems. The main difference between Java and Smalltalk is that the first is statically typed, while the second is dynamically typed, which requires a substantial different approach in the graph model definition.

Once defined the graph model used to capture the structure of the software system under study, then a code analyzer must provided to generate such a graph, starting from the software system code. With strong typed languages, this would be accomplished by parsing the source code, recognizing class and variable definitions, and generating the graph. This is the approach followed in this work to produce the graph representation of Java systems. Also, a similar approach can be found in the work of Wheeldon and Counsell [Wheeldon and Counsell, 2003]. Alternatively, the analyzer may lean on CASE tools able to reverse-engineer the code and to generate the related UML class diagram [Los Tres Amigos, 1999]. The code analyzer can then take advantage of the navigation API of the CASE tool, to explore class relationships and to build the graph. This is the approach followed in the work of Valverde and Solé [Valverde and Solé, 2003].

As regard of Smalltalk systems, no source code parser or reverse engineering is needed as we can take full advantage of the introspection of the language, to analyze the code and to capture any kind of information. On the other side, the dynamic nature of the Smalltalk language make the code less generous of information, which introduce a certain amount of uncertainty in the graph definition.

3.1.1 The Java Graph Model

Java is a static typed language. This means that coding requires to specify statically a great amount of information about variables type. In turn, this means that the code of a Java application provides the same amount of information, which can be captured across the parsing of the code itself. Thus, it is possible to define a graph model based on the type of information that can be extracted from the code.

According to our model, a Java system is represented by a graph made of a set of five different types of node and nine different types of links. Each type of node captures a specific Java entity, while each type of link captures the specific kind of an existing relationship between two Java entities.

The four types of node defined in the Java graph model are the following:

- **Class** - Representation of a Java class;
- **Interface** - Representation of a Java interface;
- **Method** - Representation of a Java class method;
- **Attribute** - Representation of an attribute. An Attribute node assumes one of the following different meanings:
  - → representation of a class attribute (instance variable);
  - → representation of an interface attribute (instance variable);
  - → representation of a method parameter;
  - → representation of a method local variable;
  - → representation of anywhere referenced variables.

The nine types of connection defined in the Java graph model are the following:

- **Contains** - Representation of the relationship between a class (interface) and its class variables; Also, it could be the representation of the relationship between a method and its local variables;
- **InnerClass** - Representation of the relationship between two classes, one defined inside the other;
- **Extends** - Representation of the inheritance relationship between classes or interfaces;
- **Function** - Representation of the relationship between a class (interface) and its methods;
- **Calls** - Representation of the relationship between two methods, one calling the other;
- **Uses** - Representation of the relationship between a method and an attribute, when the attribute is a referenced variable (both local and non local) in the body of the method;
- **Instance** - Representation of the relationship between a class and its instances;
- **Parameter** - Representation of the relationship between a method and its parameters;
- **Implements** - Representation of the relationship between a class and an interface, where the class is the implementation of the interface.

The resulting graph is an oriented graph, where an orientation is defined for each connection. Figure 3.1 gives a global view of the possible connection between the different types of node. Table 3.1 shows the meaning and orientation for each type of connection.

The built of the Java graph is performed by a smalltalk application based on the SmaCC parser generator [1]. The application performs a double parsing task. The first task parses the pure Java code and extracts all the needed information as a list of statements of the form *Relation(Entity, Entity)*, capturing the information about a specific relationship between two specific Java entities. The second task performs a new parsing process on the statement

[1]The application is developed in the Cincom VisualWorks environment (smalltalk.cincom.com). The SmaCC (Smalltalk Compiler Compiler) is a freely available parser generator for Smalltalk, developed by The Refactory, Inc. (www.refactory.com). The details of the implementation can be found in [Serra, 2001][Tuveri, 2004](Italian).

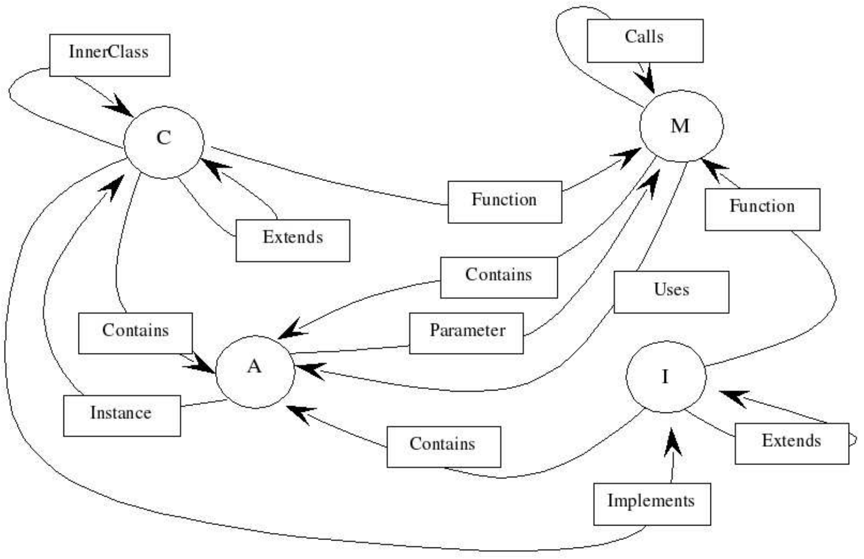

Figure 3.1: *The graph representation of a Java system. The graph is made by a set of nodes from four different types: Class (C), Interface (I), Attribute (A) and Method (M). A couple of nodes can be connected by a link from nine different types: Contains, InnerClass, Extends, Function, Calls, Uses, Instance, Parameter and Implements.*

Java Relationship	*Starting Node ⇒ Ending Node*
Contains	Class ⇒ Attribute Interface ⇒ Attribute Method ⇒ Attribute
InnerClass	Class ⇒ Class
Extends	Class (subClass) ⇒ Class (superClass) Interface ⇒ Interface
Function	Class ⇒ Method Interface ⇒ Method
Calls	Method (calling) ⇒ Method (called)
Uses	Method ⇒ Attribute
Instance	Attribute ⇒ Class
Parameter	Attribute ⇒ Method
Implements	Class ⇒ Interface

Table 3.1: *The Java system is represented as an oriented graph, where each relationship defines an orientation from a starting node of a well given type to an ending node of a well given type.*

list and definitively builds the Java graph as a persistent object. The above process leads to a graph representation displaying a large amount of information that could be translated, for instance, on the computation of many object oriented metrics. Nevertheless, we are explicitly interested on the statistical analysis of the relationships among classes, thus the application needs a one more task to perform a sort of compression of the above structure. More precisely, the application should translate each *implicit path* between two Class nodes on an *explicit path*. For example, if two Method nodes are connected with a Call link, then this relationship has to be translated in a connection between the Class nodes respectively representing the classes which those methods belong to. Following similar rules, the first complete graph is reduced in a new structure made of Class nodes only, and relationships between them. The reduced Java graph still remains an oriented graph. Thus, for a given node, it is possible to define both input and output degree, respectively as the number of out-going and in-coming links directly connected with the node itself.

This approach has a general value and could be extended to build the class graph of any software system with the only constraint of modifying the first parsing task according to the adopted programming language. Dynamic languages, such like Smalltalk, play a special role, as they do not provide much information in the code about variable typing, thus the model must be revisited and simplified. The next section provide an alternative ad hoc approach for smalltalk software systems.

3.1.2 The Smalltalk Graph Model

Similarly with the case of Java systems graph, also for the Smalltalk systems graph representation, we focused only on two kinds of relationships: inheritance and dependence. While the Smalltalk code exhaustively provides all needed information about inheritance relationship among classes, this is not the case for dependence. Typically, we say that *class A depends on class B* when class A has an instance variable whose type is B, or when class A has a method that access or defines an instance of class B.

The dependence analysis must consider that Smalltalk is not a typed language, then objects instead of variables, carry type information. This frees the programmer from declaring variable types, but embeds less information into the source code. Thus, the conditions reported above become difficult to test by a direct analysis of the code. However, we may say that the access to a variable of a given class is significant only if one or more messages are sent to that variable. That is, a dependence between two classes exists only if there is any kind of message sending among instances of those classes.

So, we define that 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 implementor, class B, the dependence link is clearly unambiguous. If the method has more than one implementor class, say n, it is not possible to ascertain with a static analysis which one is the right one. In this situation, we decided to introduce a dependency towards all implementor classes, weighting such a dependency with weight $1/n$. Inheritance is not considered in the dependency analysis. For instance, let us suppose that class C is subclass of class A, that class D is subclass of class B, and that a method of B – which is not overridden in class D – is called inside a method of A, which in turn is not overridden in class C. At run time, it may be possible that an object belonging to class C calls the method, and/or that such a method is in fact executed on an object belonging to class D. Nevertheless, in this case only a dependence between class A and class B is recorded. Note that this issue holds also for typed languages, such as Java or C++, when dynamic binding is used.

Formally, our graph is a collection of nodes:

$$G \equiv \{N_i\}_{i=1..N_c} \tag{3.1}$$

where N_c is the total number of classes within the represented software system. Thus, the graph contains one node for each system class C_i. A node N_i has a collection of links, each representing a relationship that the class represented by N_i has with other classes in the system. Links, or arcs, are pairs of nodes representing respectively the starting and the ending class:

$$L_{ij} = (N_i, N_j) \tag{3.2}$$

A link L_{ij}, carries also a weight that we indicate as $W(L_{ij})$. Now let's introduce the symbols we use for representing methods, messages and implementors:

$$M(C_i) = \{methods\,of\,the\,class\,C_i\}$$

$$S(m) = \{messages\,sent\,inside\,m\}, m \in M(C_i)$$

$$P(s) = \{implementors\,of\,message\,\mathbf{s}\}, s \in S(m)$$

$$dim(P(s)) = number\,of\,implementors\,of\,message\ \mathbf{s}$$

We define the weight of a link as the sum of related dependence, W_{dep}, and inheritance, W_{inh}, contributions:

$$W(L_{ij}) = W_{dep}(L_{ij}) + W_{inh}(L_{ij}) \tag{3.3}$$

$$W_{dep}(L_{ij}) = \sum_{m \in M(C_i)} \sum_{s \in S(m)} \frac{1}{dim(P(s))} \cdot I_s(C_j) \tag{3.4}$$

$$I_s(C_j) = \{ {}^{1\ if\ C_j \in P(s)\ (C_j\ is\ an\ implementor\ of\ s)}_{0\ otherwise} \tag{3.5}$$

$$W_{inh}(L_{ij}) = \{ {}^{1\ if\ C_j\ is\ the\ direct\ superclass\ of\ C_i}_{0\ otherwise} \tag{3.6}$$

Now we can define both the input degree, and output degree of node N_i:

$$outputDegree(N_i) = \sum_{j=1}^{N_c} W(L_{ij}) \tag{3.7}$$

$$inputDegree(N_j) = \sum_{i=1}^{N_c} W(L_{ij}) \tag{3.8}$$

The graph built in this way reflects the relationships between the classes, using in the least biased possible way the information that can be inferred by static analysis of a Smalltalk system. The input degree of a class is directly linked to the usage of this class in the system.

To perform the code analysis, in order to build the graph representation previously defined, we can take full advantage of the introspection of the Smalltalk language. The whole system is accessible from within itself, and no source code parser or reverse engineering is needed. Moreover, Smalltalk is endowed of powerful query methods able to return useful information, as 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. This is the approach followed in this work and previously in the work of Marchesi [Marchesi et al., 2003][2].

[2]To give an insight on how the dependence analysis is performed, we shortly provide some details of the implementation in Squeak (www.squeak.org). The main classes involved in the computation are *SystemNavigation*, *Class* and *CompiledMethod*. The method *allClasses* of *SystemNavigation* returns the collection of all classes in the system. For a given class, the method *selectors* returns all selectors of that class. A selector is an instance of the class Symbol and it represents the method signature. The whole method, instance of class *CompiledMethod*, can be obtained by sending the message *compiledMethodAt: aSelector* to the class. The *CompiledMethod* class implements the method *messages* which returns the collection of all messages sent inside the CompiledMethod. A message is in turn a Symbol representing the selector of the corresponding invoked method. The *SystemNavigation* class implements the method *allClassesImplementing: aMessage* which returns a collection of all classes implementing the given message. In this way, it is possible to navigate the system, building the graph described in this section.

3.2 Analysis Overview

The body of this study aims to a double objective:

> First, to analyze the degree distributions of the graph representation of an object oriented system;
>
> Second, to compute the statistical correlation between quality indicators and the graph topological properties.

The first is a wide scope objective, and targets to answer questions such like: *"What kind is the network underlining a software system? That is, what are the general statistical properties of such networks?"*

While the second one has a more focused scope, and targets to answer questions such like: *"What kind of practical information could be extracted from the network representation of a software system? For instance, could the network give us informations about the quality of the represented system?"*

3.2.1 Degree Distributions

We want to verify if the degree distributions of a graph representing an object oriented system displays a scale free effect at list in the tail, that is:

$$p_k \sim ck^{-\alpha} \text{ as k} \rightarrow \infty \tag{3.9}$$

where k is the random variable representing the node degree, c is a location parameter and α is the power shape parameter. Typically, such study is performed observing the cumulative distribution function (cdf) or the complementary cumulative distribution function (ccdf), rather than the pure distribution function.

Given the random variable X, the cumulative distribution function is given by:

$$F(x) \equiv \mathrm{P}\{X < x\} \tag{3.10}$$

then, the complementary cumulative distribution is then given by:

$$1 - F(x) \equiv \mathrm{P}\{X > x\} \tag{3.11}$$

The derivate of the cumulative distribution function is the distribution:

$$f(x) = \frac{dF(x)}{dx} \tag{3.12}$$

Note that the function given in 3.12 is also known as *density function*. In this case, the cumulative distribution is simply called distribution function and the complementary cumulative distribution is called survival distribution function.

When the distribution follows a power law, thus the ccdf still is a power law with a shape power coefficient $\beta = \alpha - 1$. Moreover, the condition:

$$1 - F(X) = \mathit{ccdf}(X) = \mathrm{P}\{X > x\} \sim ck^{-\beta} \text{as k} \to \infty \tag{3.13}$$

gives the definition of *heavy tailed distribution*. Then, the study of the ccdf is perfectly equivalent to the study of the distribution and more it provides a test of the condition 3.13.

An easy way to test the power law trend of a distribution is to plot its ccdf in a log-log scale. In fact, the log-log plot performs a transformation of a power law function into a straight line with a trend equal to the power coefficient of the original. This approach has the disadvantage that it requires the selection of a value x_0 above which the plot appears to be linear. This is the approach followed in this study. [3]

3.2.2 Correlation Analysis

We want to test if there is any statistical correlation[4] between network topological metrics, such the input and output degree, and those traditional object oriented metrics, typically used as quality indicators. For a such purpose, we have analyzed the correlation among the graph node degrees and some of the Chidamber and Kemerer metrics, some of the Martins metrics and some size metrics, such as the LOC. Note that for the Smalltalk systems, the computation of such metrics is inevitably limited by the dynamic nature of the language. More details about the selected software metrics will be provided in the next section, dedicated to presentation of the results, and in

[3] The log-log plot of the ccdf is not the only approach to test the heavy tail condition. For example, an alternative approach is given by the traditional Hill test, which nevertheless still displays the problem of the selection of critical value x_0 [Hill, 1975]. This problem was overcame by Crovella and Taqqu, which recently provided an alternative approach to test the heavy tail effect of a distribution [Crovella and Taqqu, 1999].

[4] Given two random variables X and Y, the correlation coefficient is defined as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\overline{X}\overline{Y}} \tag{3.14}$$

where cov(X,Y) is the covariance of X and Y, and $\overline{X}$ and $\overline{Y}$ are respectively the standard deviations of X and Y. It has the range $-1 \leq \rho_{X,Y} \leq 1$ and vanishes for independent variables. The extreme values reveal a linear correlation between X and Y.

Appendix A, explicitly dedicated to the description of all metrics adopted in this study.

The typical test to verify the correlation between two samples is the traditional Pearson correlation estimator. Given two sample X_i and Y_i, the Pearson coefficient is given by:

$$r = \frac{1}{1-N} \frac{\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y})}{S_X S_Y} \tag{3.15}$$

where $\overline{X}$ and $\overline{Y}$ are respectively the sample means of X_i and Y_i, and where S_X and S_Y are respectively the sample standard deviations of X_i and Y_i. The Pearson coefficient 3.15 is the simplest way to compute an estimation of the actual correlation coefficient. Nevertheless, it is based on the assumption that the random variables from which the samples have been extracted are normally distributed [Wonnacott, 1998].

When the considered variables are not normally distributed, which is the case of this study, the correlation is typically estimated by the Spearman rank correlation coefficient, given by:

$$r_s = 1 - 6\frac{\sum_{i=1}^{N} d_i^2}{N(N^2-1)} \tag{3.16}$$

where d_i is the distance between the rank of the samples [Frund and Simon, 1997]. The computations of the current study are based on the Spearman correlation coefficient, as the considered variables are far to be normally distributed.

Chapter 4

Results

4.1 Smalltalk Systems

This section presents the results of the experiment performed on some Smalltalk systems. We have considered two main Smalltalk environments: VisualWorks Cincom Smalltalk 7.2 and Squeak. Cincom VisualWorks is a professional environment to develop both commercial and non-commercial applications, while Squeak is an Open Source free Smalltalk-80 implementation.

VisualWorks has been first analyzed considering the base image containing all environment classes and all the standard parcels. Then we considered the base image with all the Store [1] parcels loaded. This allows to observe how the graph properties varies as a response of the image growth. Following this approach, we also considered some isolated parcels and categories in order to observe statistical properties of smaller graphs. For such purpose, we considered the Store parcels separately from the base image and the core classes separately from all other system classes. Moreover, we analyzed a reduced version of VisualWorks image after unloading the large part of the default parcels. Squeak has been analyzed considering the only base image, so far.

First, we will show the results of the statistical analysis displaying the Spearman correlation coefficient among the graph topological metrics and a selected group of object oriented metrics. The discussion of correlation results will provide the base for the interpretation of graph topological properties and for the interpretation of further distribution analysis. Second, for each project we will show the how the tails of the complementary cumulative distribution functions of the input and output degree are fitted by the power

[1]Store is the VisualWorks environment extension to support distributed team development and project management.

law distribution model.

4.1.1 Statistical Correlation Analysis

The following smalltalk graphs have been analyzed:

- **Graph#1**: representation of all VisualWorks base image. The graph is made of 2,345 nodes and 1,593,073 arcs.
- **Graph#2**: representation of all VisualWorks base image with Store parcels loaded. The graph is made of 2,607 nodes and 2,008,752 arcs. This is the largest graph considered in this study.
- **Graph#3**: representation of a reduced VisualWorks base image. We proceeded starting from the base image and then unloading all parcels ensuring not to break the systems image integrity. The resulting graph is made of 1,685 nodes and 947,497 arcs.
- **Graph#4**: representation of all Store classes without all other system classes. The graph is made of 210 nodes and 173,392 arcs.
- **Graph#5**: representation of Core namespace classes without all other system classes. The graph is made of 186 nodes and 186,907 arcs.
- **Graph#6**: representation of all Squeak base image classes. The graph is made of 1,798 nodes and 1,232,032 arcs.

For each systems class, metrics LOC, WMC, DIT, NOC and RFC have been computed and correlated with input and output degree. Table 4.1 and table 4.2 respectively show the Spearman correlation coefficient computed first between input degree and the above metrics and second between output degree and the same metrics. The descriptive statistic of each analyzed system is provided in Appendix B.

	LOC	**WMC**	**DIT**	**NOC**	**RFC**
Graph#1	0.7103	0.8394	-0.2666	0.3552	0.7466
Graph#2	0.7059	0.8397	-0.2730	0.3545	0.7465
Graph#3	0.6902	0.8494	-0.2870	0.3538	0.7228
Graph#4	0.7459	0.9058	-0.2675	0.3361	0.8085
Graph#5	0.8170	0.8737	-0.0797	0.3463	0.8259
Graph#6	0.7050	0.8115	-0.2494	0.3376	0.6933

Table 4.1: *Statistical correlation between Input Degree and software metrics.*

	LOC	WMC	DIT	NOC	RFC
Graph#1	0.8615	0.8816	-0.0674	0.2283	0.9835
Graph#2	0.8583	0.8816	-0.0764	0.2187	0.9827
Graph#3	0.6356	0.8706	-0.0723	0.2242	0.9795
Graph#4	0.8307	0.8719	-0.1107	0.0990	0.9753
Graph#5	0.9541	0.9563	-0.1625	0.2204	0.9848
Graph#6	0.9056	0.8885	-0.0401	0.2250	0.9827

Table 4.2: *Statistical correlation between Output Degree and software metrics.*

Results show that both Input Degree and Output Degree are strongly correlated with the computed size metrics (LOC and WMC) and with the RFC metric, which is an indicator of the coupling between class. Rather, no correlation emerges with the inheritance metrics (DIT and NOC).

The Output Degree, that is the number outgoing links directly connected with a node, gives a measure of how many messages are sent from within the methods of the class represented by such node, to other classes in the system. Thus, the Output Degree is an indicator of how a certain class depends on other external classes while providing its services. The correlation between the Output Degree and the class size indicators, namely the class LOC and the WMC, shows that big classes are more likely to depend on external classes. Specifically, the WMC metric, which in fact corresponds to the number of methods of a class, is reasonably related with the number of outgoing links, as each outgoing link represents a message sent by a method of the represented class. Moreover, the correlation with RFC metric shows that classes with a great number of outgoing links are more likely to be strongly coupled with external classes. Thus, the Output Degree, which for construction is an indicator of the dependence of a class with other external classes, in fact also brings information about the class size and about the class coupling. Note that the Output Degree also takes account of the inheritance relations, which is usually considered orthogonal with respect to dependence and thus it is typically not accounted for while measuring coupling. Nevertheless, since all classes have a single superclass in Smalltalk, the link representing this relationship adds one to the outgoing links value of all classes, and is therefore immaterial. This explains the low values of the correlation coefficient among Output Degree and inheritance metrics.

The Input Degree, that is the number of incoming links directly connected with a node, gives a measure of how many messages a certain class receives by other classes of the system. Thus, Input Degree is an indicator of how much a certain class is referenced by other classes, while they provide their services.

Here the interesting result is represented by the correlation of the Input Degree with the RFC metric. This suggests a reasonable scenarios, where a class largely used by other classes in the system, is more likely to have a large response set, that is the number of methods that can be potentially executed in response of a received message. Then, the Input Degree provide a better indicator of the class coupling, which is counted in a double direction, both from the environment to the considered class and from the considered class to the external environment. Inheritance is simply not accounted by the construction of the Input Degree of a class, which determines low values of the correlation coefficient among the Input Degree and the inheritance metrics.

The high values obtained for the correlation also need a certain amount of attention and warning in their interpretation. The main problem consists with the fact that all metrics have been directly computed upon the code using the same mechanisms adopted to build the graph (see Appendix A). The problem can be partially overcame comparing the above results with the results obtained for the Java systems and presented in section 4.2, where the graph analysis is completely independent from the software metrics computation.

4.1.2 Distribution Analysis

The first set of plots given in Fig.4.1 display the frequency distributions of *Input Degree* and *Output Degree* respectively, both in a standard and in a log-log plot, for the graph representation of the base image of VisualWorks (Graph#1). These plots do not give much information about the best theoretical underlining distribution. Anyway, they give a first idea of the general shape. As we can see, there is a large number of classes having few links and a small number classes having a large number of links. Moreover, there is no "typical" mean value, which represents a first shift from the random graph Poisson model to the scale-free mode.

Following the approach described in section 3.2.1, we will provide the analysis of the complementary cumulative distribution functions (ccdf) both of input and output degree. We remind that, given a random variable X the ccdf gives the probability of such variable to be greater than the current value, that is:

$$1 - F(X) = ccdf(X) = \mathrm{P}\{X > x\} \tag{4.1}$$

All the complementary cumulative distribution functions have been plotted in a log-log scale. The log-log plot performs a transformation of the power

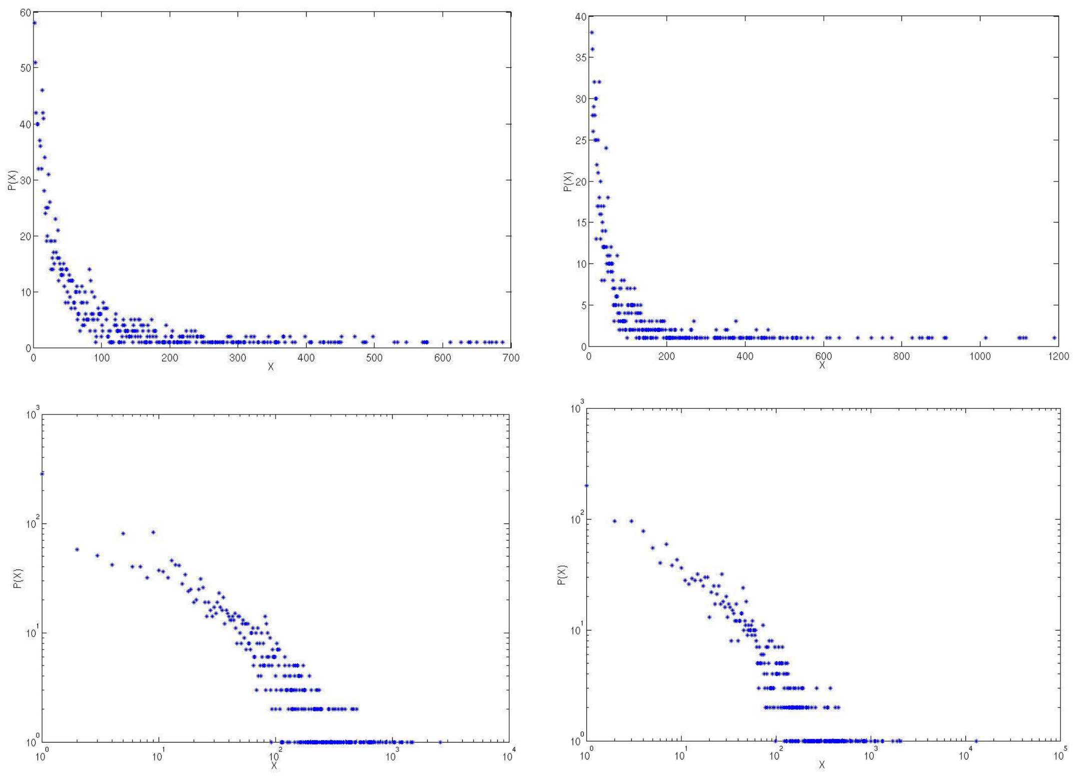

Figure 4.1: *Density functions for both the Input Degree and Output Degree for the Visual Works graph (Graph#1). Starting from the high plot on the left and moving clockwise, the first and the second windows display respectively the output degree and the input degree density in a standard plot; the third and the fourth display the same functions in a log-log scale plot. The above plots don't give much information about the underlining theoretical distribution. Nevertheless, above all the log-log scale plots suggest a slow decreasing trend. Moreover, it is clearly shown how there is no "typical" mean value, which represents a first shift from the random graph Poisson model to the scale-free model.*

law function into a straight line function, with slope equal to the power exponent of the original curve. This allows to visually better verify the presence of a power law and to estimate the power coefficient.

Plots displays how the ccdf tails are in fact well fitted by a straight line in a log-log scale, which suggests the existence of the heavy tail effect in such distributions. Figures 4.2 and 4.3 display respectively the Input Degree and Output Degree ccdf tails for each analyzed Samlltalk system. Table

4.3 shows the power exponent α of the power law distribution which best fits the empirical distributions of each system graph. Note that α represent the power coefficient of the distribution function, while the complementary cumulative distribution coefficient is given by $\beta = \alpha - 1$

The power exponent α of the distributions is quite close to the theoretical value ($\alpha = 3$) obtained by Barabasi and it is coherent to the empirical values ($2 < \alpha < 4$) obtained for many real networks. An higher exponent denotes a tail decreasing quicker. A small exponent denotes a "fatter" tail, that is a bigger deviation from the behavior of a Gauss or Poisson distribution.

	Input Degree	**Output Degree**
Graph#1	$\alpha = 2.5689$	$\alpha = 2.8554$
Graph#2	$\alpha = 2.5877$	$\alpha = 2.8289$
Graph#3	$\alpha = 2.5769$	$\alpha = 2.9218$
Graph#4	$\alpha = 3.7320$	$\alpha = 2.4974$
Graph#5	$\alpha = 2.1779$	$\alpha = 2.9254$
Graph#6	$\alpha = 2.2835$	$\alpha = 2.5946$

Table 4.3: *Power Coefficients. Power coefficient of the Distribution Function for each analyzed smalltalk system. Note that this is the α parameter previously described in section 3.2.1. The relative CCDF power coefficient is given by: $\beta = \alpha - 1$.*

Despite this is a general result, which is in fact shared by various kind of different real networks, our main interest is to interpret its meaning in the specific context of software systems. The number of outgoing links of a node is a measure of how many messages are sent from within the methods of the corresponding class. A high value of the number of outgoing links denotes that the class performs many message calls from within its methods. This can be considered also as a measure of the coupling between classes. Thus, a power law distribution with fat tails is a clue that the system is characterized by a certain number of classes with an high level of coupling, while the bulk of the classes tends to be much less coupled. This interpretation is supported by other previous studies. For example, Chidamber and Kemerer [Chidamber at al, 1994] and Basili [Basili et al., 1996] found similar distributions for coupling metrics.

Since the Output Degree of a node is not divided by the number of methods of the related class, it is reasonably proportional to the number and size of methods of the class, as suggested by the previous correlation analysis. Consequently, the power law behavior of the Output Degree distribution surely reflects the distribution of class sizes, and then suggests that, when

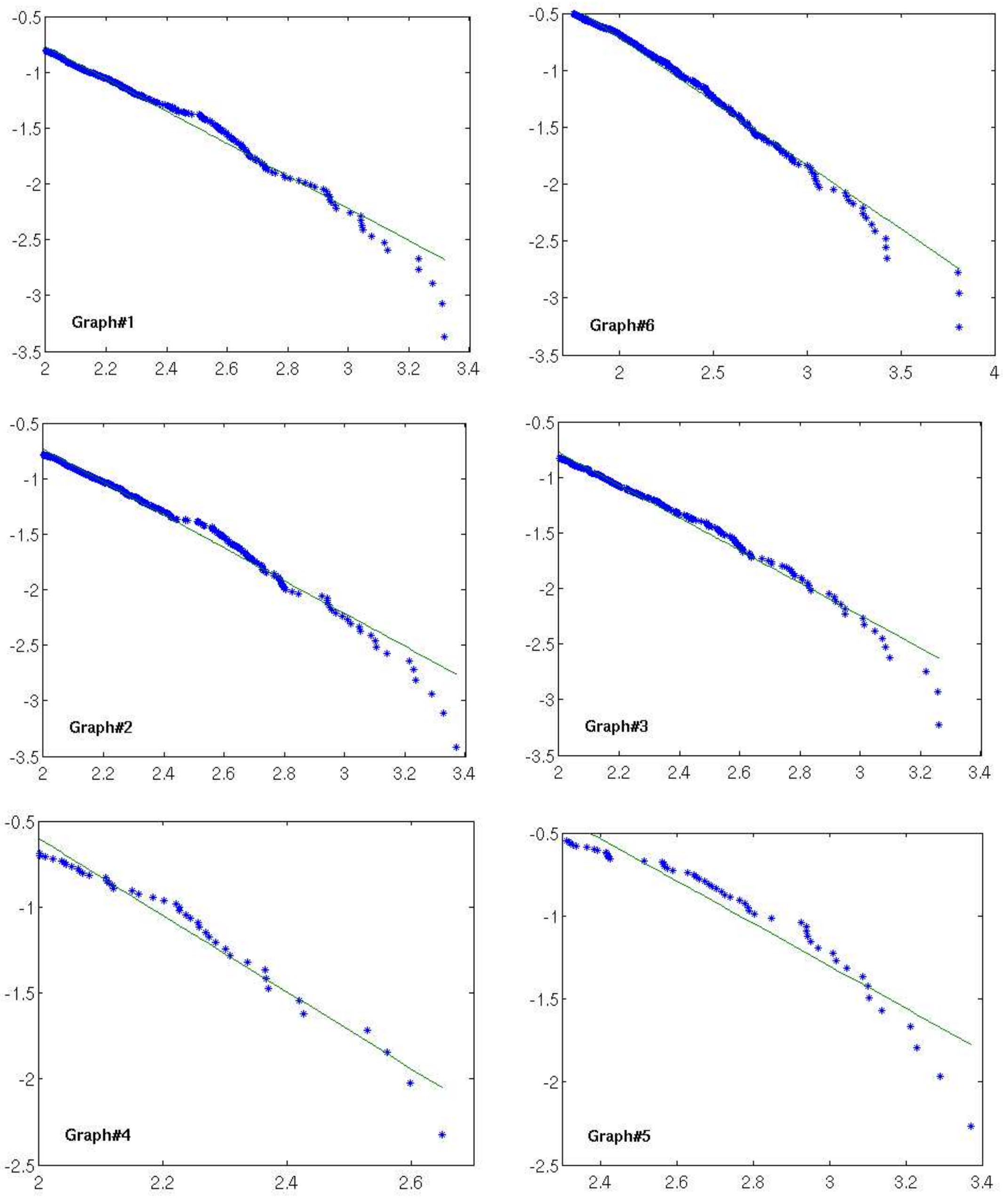

Figure 4.2: *Input Degree CCDFs. Distributions tails plotted for each Smalltalk system: Graph#1, Graph#6, Graph#2, Graph#3, Graph#4 and Graph#5. The linear trend in a log scale provides the evidence of the presence of the scale-free effect in all the analyzed networks.*

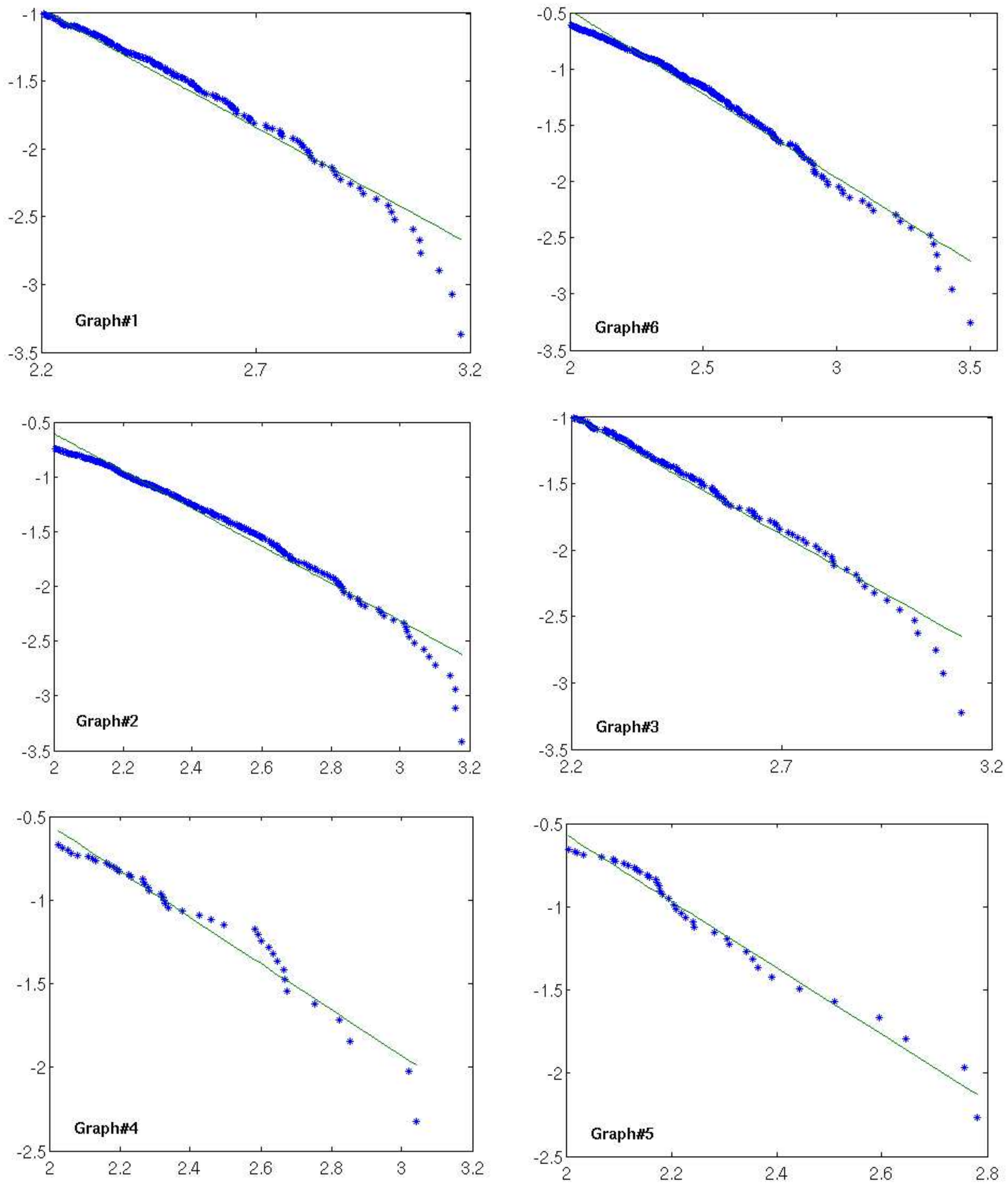

Figure 4.3: *Output Degree CCDFs. Distributions tails plotted for each Smalltalk system: Graph#1, Graph#6, Graph#2, Graph#3, Graph#4 and Graph#5. The linear trend in a log scale provides the evidence of the presence of the scale-free effect in all the analyzed networks.*

a new method is added to the system, it is more likely that it belongs to a class that already has many methods.

Note that both high coupling and large classes are not considered good object oriented programming style. A good object oriented system should be composed of many cohesive classes, at different detail levels, each one focusing on a single task, and endowed with a few, short methods. The empirical analysis of the Smalltalk system shows that this is not the case, as displayed by the distribution of the number of outgoing links decaying as a power law.

The other main indicator we examined is the input degree, which gives a measure of how a certain class is referenced by other classes in the system. The shape of the distribution is due to the fact that in Smalltalk system, there are “service” classes that are used by almost all other classes. For the largest analyzed images holding all the system classes (Graph#1, Graph#2, Graph#3 and Graph#6), classes such as **Object**, the **Collection** and the **Magnitude** hierarchies, come immediately to mind. Nevertheless, the fact that this behavior is still displayed by the small images (Graph#4 and Graph#5), suggests that this is a more general effect.

There are also classes which are fairly used, both in the system and in specific parcels, while most other classes are rarely used. The usage rate is certainly not random. This behavior suggests the hypothesis that when writing a new method the probability to send a message implemented in another class is roughly proportional to the number of sending of that message in the system. More precisely, such probability seems to be more than proportional to this number, as the coefficient values smaller than 3 suggest.

4.1.3 More Distributions

This section presents some results relative to an extra analysis performed on the VisualWorks system image represented by the Graph#1. We will show that the scale free effect, displayed by the Input Degree and Output Degree indicators, is also displayed by other software properties not requiring a graph modeling. The results presented here are related with the work of Concas and his colleagues [Concas et al., 2005], where the following results and discussion have been also extended on two Java systems, namely the Eclipse[2] project and the JDK[3] project.

Figures 4.5 and 4.6 are relative to measures performed at class level, while plots in figure 4.7 are relative to measures performed at method level. The

[2]An Open Source free IDE for Java development (http://www.eclipse.org).

[3]The Java Development Kit provided by Sun Microsystems (http://java.sun.com)

plot in figure 4.4 shows the best power law fit for the distribution tails respectively of the NoC (Number of Children) and of the number of all subclasses. The plot in figure 4.5 shows the distribution tails of the class size both in terms of line of code (LOC) and of number of methods (NoM). As one can see, such distributions is better fitted by a lognormal function, rather than a power law. In some way, this still implies the presence of a fat tail effect [4], then showing that the system is made of a large number of small size classes, while there is a certain number of classes having a great length. This feeds the suspect that the degree distributions reflects in fact this behavior, as the high correlation coefficient between size metrics and degree metrics warn. This idea is finally confirmed by the plot of figure 4.6, which represents perhaps the most interesting result. The plot show the distribution of the Output Degree scaled on the class size in terms of LOC. Surprisingly, the trend is fully fitted by a Gaussian shape, rather than a power law. The meaning is that the pure Output Degree of a class, once cleaned of the class size information, is randomly distributed according to a Gaussian law, then following the traditional Random Graph model rather than the modern scale free network model.

Metric	**Power Coefficient**
Method LOC	$\alpha = 3.0967$
Calls to Method	$\alpha = 2.0135$
Method Implementors	$\alpha = 2.5693$
Calls Vs. Implementors	$\alpha = 2.1993$
Class NoC	$\alpha = 2.0850$
Class Number of All Subclasses	$\alpha = 2.1573$

Table 4.4: *Power Coefficients. Power coefficients of the Distribution Function for each metric computed on all the Smalltalk systems.*

The last set of plots in figure 4.7 shows the analysis of the distributions of some method indicators. Namely, the method size in terms of line of code; the number of implementors of a given method selector, that is how many times a given selector name has been used within the systems by different

[4]The formal definition of heavy tailed distribution is the one given in 3.13. Thus, is not *formally* correct to say that a lognormal distribution displays the heavy tail effect. Moreover, the lognormal shape is not free of scale, as it presents a peak similar to the gaussian distribution. However, the tail of the lognormal function still appear to be a straight line if plotted on a log scale. This is the main reason why, in this context, this kind of trend is considered more closer to the power law rather to the gaussian behavior. More details about the lognormal distribution and a wide view of other distributions could be find in the ARL Report indicated in [Saucier, 2000].

classes; the number of calls of a given selector performed within the system by other methods, that is the number of senders of a given message selector; the number of calls of a given method selector scaled on its number of senders. Each metric distribution is again characterized by fat tails following a power law.

Table 4.4 provides the power coefficients for each analyzed case. Note that all values remain around the theoretical value and then close to those obtained for other previous studies relative to both software networks and non-software networks.

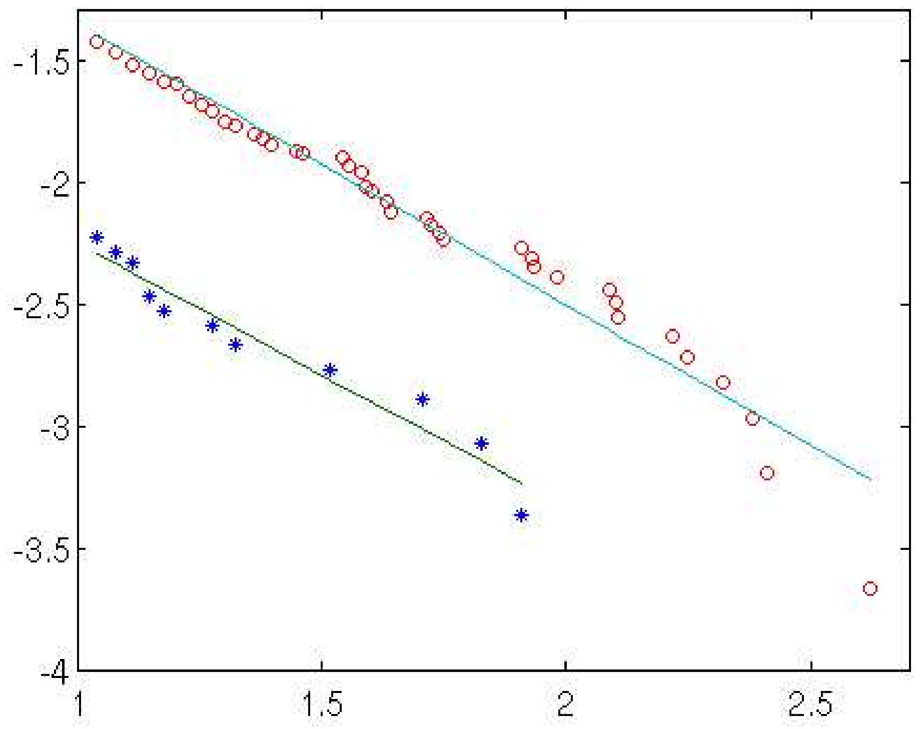

Figure 4.4: *CCDF of the inheritance metrics for the base image of the VisualWorks environment (Graph#1). Two metrics are plotted: Number of Cildren (*) and Number of All Subclasses (o).*

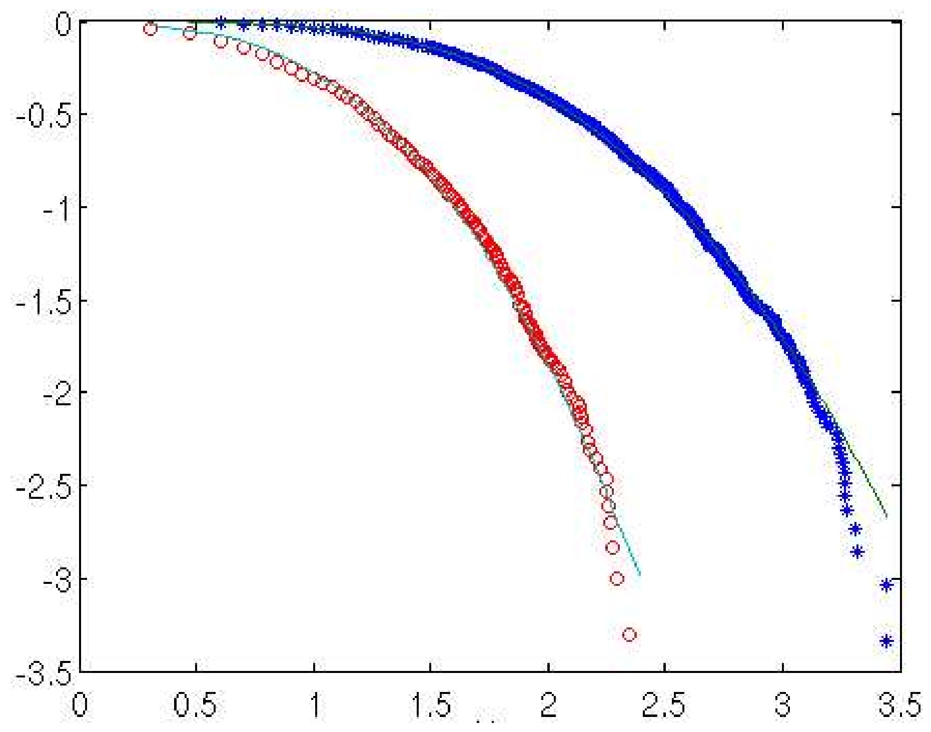

Figure 4.5: *Class Size Metrics. CCDF tails of the system class size in terms of class LOC (o) and in terms of Number of Methods per class (*), for the base image of the VisualWorks environment (Graph#1). Both plots show that the size distribution of the VisualWorks system is well fitted more by a lognormal trend rather than a power law. The lognormal distribution formally differs from the heavy tailed distribution and does not display the scale free effect. However, if plotted on a log scale it still appears, at least in the tail, as a straight line. Thus, in this context, the lognormal behavior still can be considered close to a power law distribution.*

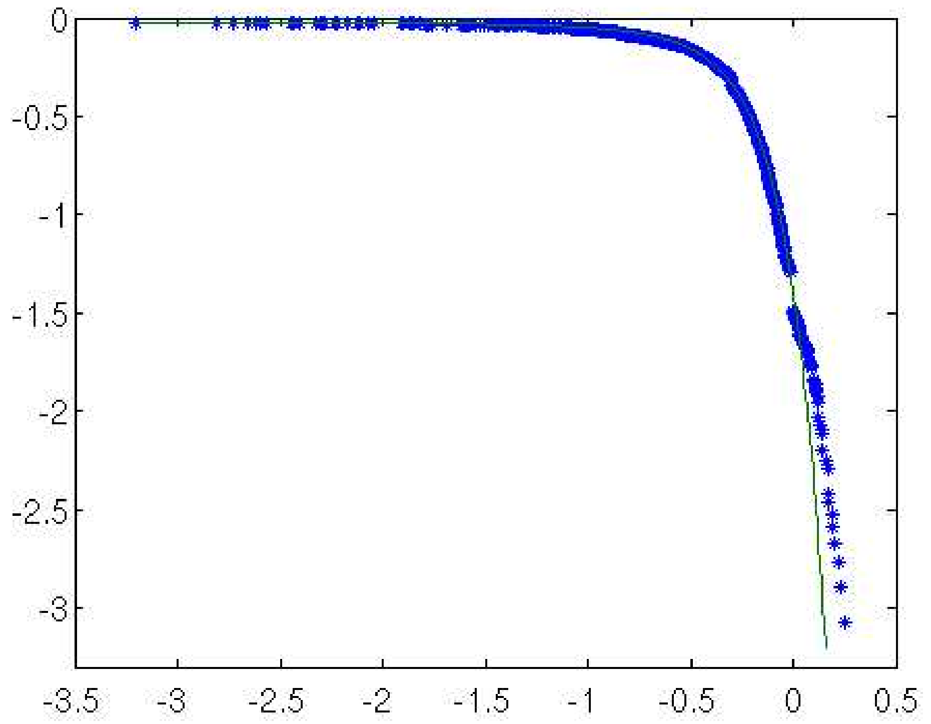

Figure 4.6: *CCDF of the Output Degree normalized by the class size in terms of class LOC for the base image of the VisualWorks environment (Graph#1). Surprisingly, this indicator is fully fitted by a normal Gaussian distribution, rather than a power law or a lognormal. This means that the power-law or lognormal behavior found in the output links distribution here and by other authors in their works is in artifact. They in fact measured the lognormal or power law distribution of class sizes. The use of external services by segments of code of the same size looks fairly random and independent. So far, this is the only normal distribution found in our and other related works about software systems networks.*

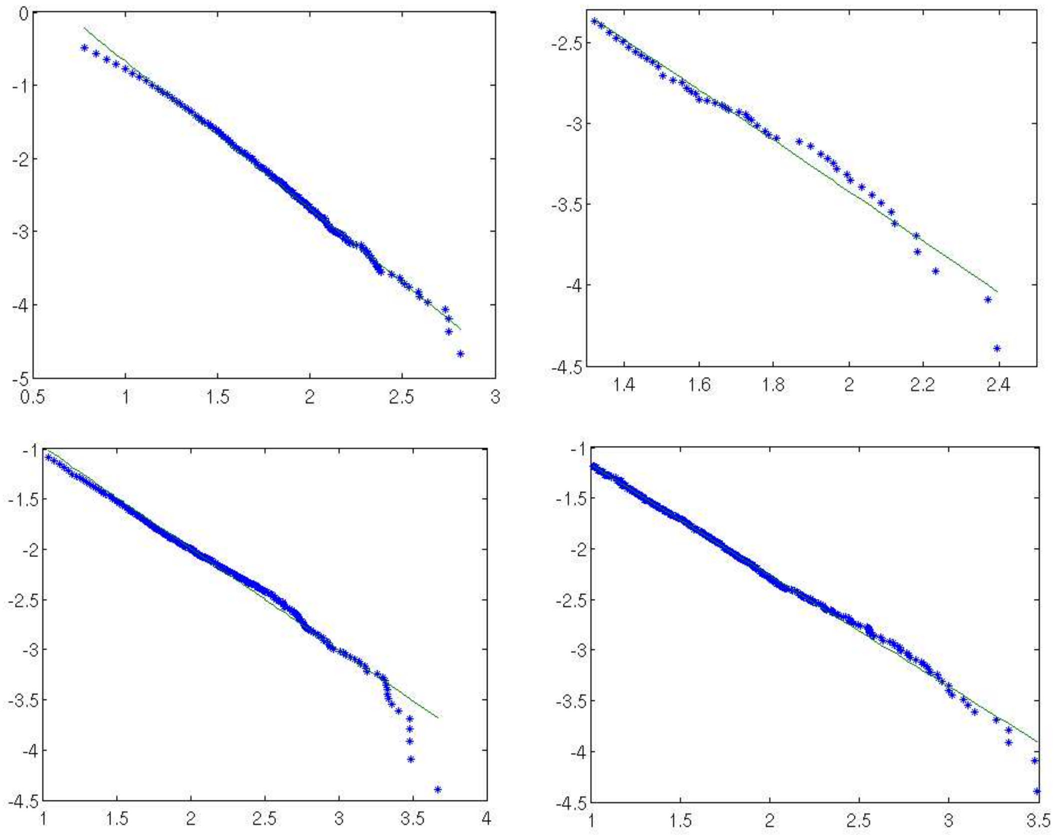

Figure 4.7: *Method Metrics. CCDF tails of some method metrics for the base image of the VisualWorks environment (Graph#1). Staring from the top left and moving clockwise: Method Size, Method Names, Method Calls, Method Calls vs. Number of Implementors. The size of a method is given in terms of number of line of non-commented code. The method name metric gives the number of times a certain selector name is implemented in the systems, that is the number of implementors of a given selector. The Method Calls metric gives the number of times a given selector is referenced within the system, that is the number of all senders of a given selector. The last gives the rank of the method calls up the number of implementors for each method in the system.*

4.2 Java Systems

This section presents the results of the experiment performed on two Java systems, namely the Apache Ant system and the Apache Jakarta Tomcat system. The first is a Java-based build tool, while the second is an application server for Java web components.

Similarly to what has been made on the Smalltalk systems, for each systems we first built the graph representation, according to the approach described in section 3.1.1, and then we performed the statistical correlation among the node degrees and some object oriented metrics. Second, we performed the analysis of the degree distributions showing how the tails of these distributions display the scale-free effect.

4.2.1 Statistical Correlation Analysis

The following Java graphs have been analyzed:

- **Ant**: The system is made up of 901 class, which is in turn the number of nodes of the relative graph representation. The distribution analysis has been performed on such data set, while the correlation analysis has been computed on a subset of 611 classes.

- **Tomcat**: The system is made up of 677 class, which is in turn the number of nodes of the relative graph representation. The distribution analysis has been performed on such data set, while the correlation analysis has been computed on a subset of 243 classes.

Metrics LOC, Ce, LCOM-CK, LCOM-HS and WMC have been computed and correlated with input and output degree. Table 4.5 shows the Spearman correlation coefficient computed first between input degree and the above metrics and second between output degree and the same metrics. The descriptive statistic of each analyzed system is provided in Appendix B.

	LOC	**Ce**	**LCOM-CK**	**LCOM-HS**	**WMC**
Ant - InDeg	-0.0056	0.0936	0.1588	0.0929	0.1298
Ant - OutDeg	0.6366	0.8125	0.4325	0.4174	0.7483
Tomcat - InDeg	0.0320	0.1048	0.2232	0.2147	0.1856
Tomcat - OutDeg	0.6800	0.8093	0.4703	0.5799	0.7647

Table 4.5: *Statistical correlation of Input Degree and Output Degree with software metrics.*

The correlation analysis performed on the Java systems differs from the previous analysis made on the Smalltalk systems, for two main reasons:

1. First, the set of selected metrics is different between the two case study. For the Smalltalk case, we have considered two metrics related with the inheritance mechanism, namely the DIT and NOC metrics. Inheritance resulted to be statistically uncorrelated with the graph properties and this is reasonable as the graph brings information about dependence in terms of message sending rather than in therms of inheritance. Similar observations can be done on the Java graph model, where inheritance is still not significant as a consequence of the single inheritance mechanism. In the Java analysis the only metric that in some way takes account of the inheritance is the Ce metric, which nevertheless is a general coupling metric. Differently of the Smalltalk case, we have computed two cohesiveness metric, namely the LCOM-CK and the LCOM-HS. The dynamic nature of the Smalltalk language make these metric difficult to practically define and implement, which is not the case for Java, that brings in the code a great amount of information about types. For both Smalltalk and Java, we have computed the same size metrics, namely the LOC and the WMC.

2. Second, the computation approach of software measures is different between the two case study. More precisely, the metric computation on the Smalltalk systems has been made directly on the code relying on the system self inspecting ability. It is important to note that the same mechanism and quite the same peaces of code have been used to build the graph representation (see Appendix A). This reasonably affects the statistical correlation computation among the software metrics and the graph topological indicators. This is not the case of the Java systems, where the software metrics computation and the graph analysis are independently computed relying on two different tools.

The correlation analysis shows a fundamental difference if compared with the Smalltalk case study. As we can see in table 4.5, while the Output Degree is still correlated with about all computed metrics, the Input Degree is no more correlated with them. This confirm that the Output Degree brings in fact information about the class size and that a large class with a great number of methods is more likely to depend on external classes. This line is still enforced by the correlation of the Output Degree with the cohesiveness metrics, which in fact suggests a scenarios where classes with a great number of outgoing links are more likely to be poorly cohesive. Moreover, the correlation with the Ce metric, which is a coupling indicator warns that classes with a large number of outgoing links are likely to be also heavily coupled with external classes.

The Input Degree remains difficult to interpret in terms of software quality, as a consequence of the low correlation values found. Such results also are in great contrast with previous Smalltalk analysis, that consequently needs more caution in its interpretation.

4.2.2 Distribution Analysis

Following the approach described in section 3.2.1 and then in the previous Smalltalk anlysis, we will provide the complementary cumulative distribution functions (ccdf) both of Input and Output Degree. Once more, all the complementary cumulative distribution functions have been plotted in a log-log scale to better verify the presence of a power law and to easily estimate the power coefficient.

Plots displays that similarly to the previous analysis the ccdf tails are in fact well fitted by a straight line in a log-log scale, which confirms the existence of the heavy tail effect in such distributions. Figure 4.2.2 displays both the Input Degree and Output Degree ccdf tails for each analyzed Java system, while Table 4.6 shows the power exponent α of the power law distribution which best fits the empirical distributions of each system graph.

Results are quite close to those provided by the Smalltalk case study. Note that the power exponent α of the distributions is still close to the theoretical value ($\alpha = 3$) obtained by Barabasi and to the empirical values ($2 < \alpha < 4$) obtained for many real networks.

	Input Degree	**Output Degree**
Ant	$\alpha = 2.2560$	$\alpha = 3.4135$
Tomcat	$\alpha = 2.4969$	$\alpha = 2.5972$

Table 4.6: *Power Coefficients. Power coefficients of the Distribution Function for each analyzed Java system. Note that this is the α parameter previously described in section 3.2.1. The relative CCDF power coefficient is given by: $\beta = \alpha - 1$*

For the analyzed Java systems, we can replay the same observations made for the Smalltalk systems.

With respect to the Output Degree, the power law distribution shows that the system is characterized by a certain number of classes with an high level of coupling, while the great part of them tends to be much less coupled. This interpretation is here enforced by the high correlation found between

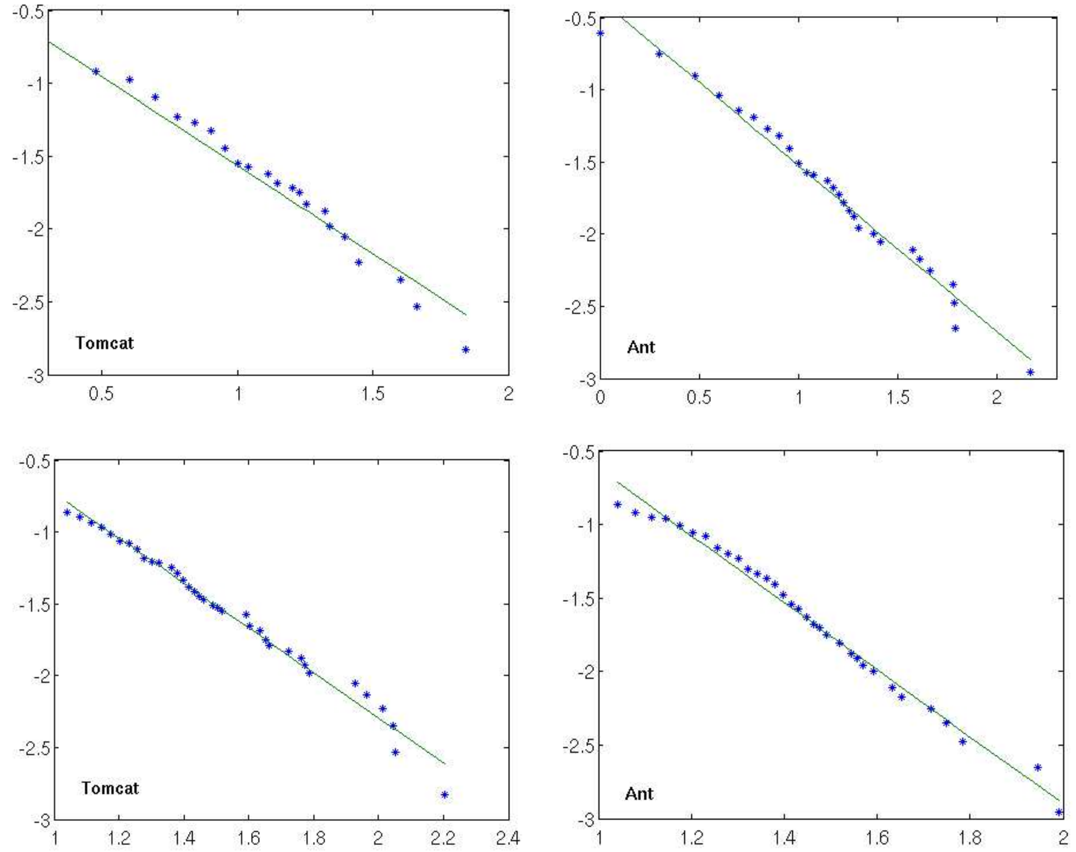

Figure 4.8: *Java Distributions. Input Degree and Output Degree distributions tails for each analyzed Java system: Tomcat Input Degree and Ant Input Degree on the top; Tomcat Output Degree and Ant Output Degree on the bottom.*

the Output Degree and the Ce metric, which is a coupling indicator. While the correlation with the size metrics LOC and WMC, again suggest that the Output Degree power law distribution reasonably reflects the distribution of class sizes and that when a new method is added to the system, it is likely that it belongs to a class that already has many methods.

The Input Degree gives a measure of how a certain class is referenced by other classes in the system. The distribution trend is due to the fact that also in the Java system, there are "service" classes that are used by almost all other classes, while there are also classes which are fairly used and many classes that are rarely used.

Chapter 5

Conclusions and Further Works

Modeling deals with providing an abstraction of any reality aspect, in order to strip away needless details while focusing on needed ones. Measuring and modeling are strictly related one to each other. A measure definition needs in fact to be associated with a model, which in turn provides the abstraction of the entities and attributes to be measured. In this work we have presented a model representing a general object oriented software system. The model definition is inspired by the Complex Network and Random Graphs theories. According to such model, an object oriented system is made of a set of entities, such as classes, methods and attributes, related across specific relationships. First, the model can be useful to provide a base for the design and implementation of tools for both the definition and extraction of many software metrics. Second and more important, the same model provides a new perspective and view from which to analyze object oriented software system general properties. This is the perspective of the Complex Network, where a real system and the relationship among its subparts are analyzed on a statistical base. This study has been mainly focusing on the second aspect.

5.1 Conclusions

This work started with a double goal:

- Analyze the general statistical properties of an object oriented system in terms of the degree distribution of the underlining network representation;

- Interpret the network topological informations in terms on software quality;

We model and analyze six different Smalltalk systems and two different Java systems. For each system, we built the network representation using ad-hoc designed parsing tools, we analyzed the degrees statistical distributions of the relative network representation and we performed the statistical correlation among the topological network properties and some software quality indicators.

We systematically found power law heavy tailed distributions for both input and output degree, denoting that the programming activity in no way can be simply modeled as a random addition of dependent increments, but exhibits strong dependencies on what has been already developed. We tried to explain the reasons behind the power law behavior of the distributions studied, discussing how a program is built and which dependencies may arise. We also discussed how quality measures of object oriented systems can be linked with the behavior of the studied distributions. This could be a starting point to devise a set of quality metrics based on overall statistics behavior of large systems.

We explained why the distribution of output links in the class graph, which represents the number of other classes used by a single class has been consistently found by us, and by other authors in other previous studies, as power law or sometimes lognormal. This behavior is directly linked to the power law or lognrmal distribution of the class sizes - the bigger the code of a class, the more the dependencies on other classes in the system. Normalizing the output links number on the class size, the resulting output degree distribution becomes very close to a Gaussian distribution. Interestingly, this is the only case of quasi-Gaussian distribution we found in our and other related studies. The idea of the existence of a strong relation among the output degree and the class size is also confirmed and enforced by our statistical correlation analysis. The output degree has been also found to be strongly correlated with other quality indicators, such like coupling and cohesion metrics.

The input degree remains less clear in its interpretation. While it is not difficult to give a qualitative interpretation of such indicator, it results difficult to find a more quantitative interpretation in terms of quality.

5.2 Further Works

The study presented here suggests new ideas and scenarios for further works. First of all, it could be useful to repeat the analysis on other software systems in order to confirm the above conclusions and eventually overcome any doubt. This could be achieved relying on the large amount of code maintained by the Open Source community across the Internet.

Going further, it could be useful to perform more correlation analysis considering other software metrics not necessarily related with quality, but rather accounting different software product and process properties. Also, complex networks provide a new perspective to evaluate the evolution of a software system in terms of the underlining graph evolution. Moreover, this gives the opportunity to develop new models of networks having power law degree distribution, as an alternative to the Barabasi preferential attachment model. This could the starting point to evaluate how the network model attributes varies for examples as a response of the different development methodology used to produce the represented software system.

Last, following the state of the art of the Complex Network theory, it could be interesting to evaluate the presence of self similar structures inside the software networks, and then provide an interpretation in terms of software evolution.

Appendix A

Metrics

This section describes all metrics computed in this study, both for Smalltalk and Java systems. It also explains some implementation detail above all for the part relative to Smalltalk. The list of all computed metric is the following:

- **LOC - Lines Of Code** - Gives a code size measure. There is not a unified definition of such metric, but usually the name LOC is intended to give the number of uncommented lines, where CLOC gives the number of commented lines. Nevertheless, note that the meaning of *line of code* itself is not unambiguous. For examples, it sometimes gives the actual number of carriage return characters, while sometimes it is intended as the number of statements, where multiple lines statements are counted once. This last definition comes from the Hewlett-Packard where a line of code has been in fact formally defined as: *Any statement in the program except for blank lines and comments* [Grady and Caswell, 1987]. The Hewlett-Packard definition of a line of code is actually the most widely used. In a sense, valuable length information is lost while using this definition, for example when deciding how much computer storage is required. Nevertheless, it is a reasonable and useful metric definition for instance if we are interested in estimate the effort from the point of view of the productivity assessment [Fenton, 1997].

  In this study we counted the actual number of carriage return character, both in Java and in Smalltalk systems. For the Java systems we computed the LOC metric with the TEAMINABOX Eclipse Metrics Plugin (http://www.teaminabox.co.uk). For the smalltalk systems we based the computation on some system methods allowing to perform a direct parsing code. More precisely, the computation is based on the method *sourceCodeAt: messageSelector* of the class *Behavior*, which answers the string corresponding to the source code for the argument, namely a method selector. Thus, a peace of code like the following:

```
theClass sourceCodeAt: messageSelector
```

returns a string containing the code which implements the method of the class *theClass* and with selector *messageSelector*. The the returned string could be parsed to count special characters such like the carriage return. Note that while the Java metric counts also the length of the header of the method, this is not counted in the Smalltalk case.

- **WMC - Weighted Methods per Class** - Gives a code size measure both in terms of length and complexity. The formal definition of the metric has been given in equation 1.1. Each method complexity is usually computed as the McCabe Ciclomatic complexity [McCabe, 1976] or alternatively is considered unitary then giving the number of methods for a class, which is a pure class length measure. The number of methods and their complexity are predictors of the effort required to develop and maintain the class. The larger the number of methods, the greater the potential impact on children classes, as they inherit all methods of parent classes. Moreover, a class with a large number of methods are likely to be application specific, limiting the possibility of reuse.

  In this study the metric has been computed with unitary complexity both for Java and for Smalltalk. For the Java systems we computed the WMC metric with the TEAMINABOX Eclipse Metrics Plugin. For the smalltalk we computed the metric directly inspecting the code from the environment. The computation is based on the method *selectors* of the class *Behavior*, which return the collection of all selectors of the class receiving the message. Thus, the required measure is simply given by a peace of code like the following:

  ```
  theClass selectors size
  ```

- **DIT - Depth of Inheritance Tree** - Gives a measure of the use of inheritance mechanism. The formal definition of the metric has been given in section 1.3.1. The DIT metric gives the number of ancestors of a given class, in cases involving multiple inheritance, the DIT will be the maximum length from the class to the root of the tree. This metric has been computed only on the Smalltalk systems. The computation is easily provided by the method *allSuperclasses* of the class *Behavior*, which return a collection of all the classes in the hierarchy of the class receiving the message. Thus, the required measure is simply given by a peace of code like the following:

```
theClass allSuperclasses size
```

- **NOC - Number of Children** - An other measure of the inheritance. NOC is simply the number of direct subclasses of a given class. Like the DIT metric, it has been computed only on the Smalltalk systems. The computation is easily provided by the method *subclasses* of the class *Behavior*, which return a collection of all the direct child classes of the class receiving the message. Thus, the required measure is simply given by a peace of code like the following:

  ```
  theClass subclasses size
  ```

- **RFC - Response for a Class** - This is a measure of the communication of the class with other classes in the system. The greater the number of methods that can be invoked by a class, the larger the complexity of the class. RFC gives a measure of the effort required for maintain a class in terms of testing time. The formal definition of RFC has been given in equation 1.4. This metric has been computed only on the Smalltalk systems. The computation is hardly influenced by the dynamic nature of the smalltalk language. Thus, the measure results to be not fully coherent with the original definition given by Chidember and Kemerer. More precisely, as there is no way to statically detect the class implementing a given method selector, methods having the same name, but potentially implemented within different classes, are counted only once. Thus, the required measure is computed by a peace of code like the following:

  ```
  | set |
  set := Set new.
  aClass selectors do:
    [:selector |
     set add: selector.
     set addAll: (aClass compiledMethodAt: selector) messages].
  ^set size
  ```

It is important to note that the above code is quite close to the code which implements the building of the smalltalk graph. Thus, it is reasonable that such metric will be heavily correlated with the Input Degree and Output Degree of a class.

- **Ce - Efferent Coupling** - Gives a measure of the coupling between classes. This metric has been proposed by Robert Martin as a part of a suite of metrics known as Martin Package [Martin, 1994].

  The metric definition is based on the idea of the *Class Category*. Referring to the original paper, a Class Category is a group of highly cohesive classes that obey the following three rules:

  1. The classes within a category are closed together against any force of change. This means that if one class must change, all of the classes within the category are likely to change. If any of the classes are open to a certain kind of change, they are all open to that kind of change.
  2. The classes within a category are reused together. They are strongly interdependent and cannot be separated from each other. Thus if any attempt is made to reuse one class within the category, all the other classes must be reused with it.
  3. The classes within a category share some common function or achieve some common goal.

  These three rules are listed in order of their importance. Rule 3 can be sacrificed for rule 2 which can, in turn, be sacrificed for rule 1.

  If categories are to be reused, they must also be released and given release numbers. If this were not the case, reusers would not be able to rely upon the stability of the reused categories since the authors might change it at any time. Thus the authors must provide releases of their categories and identify them with release numbers so that reusers can be assured that they can have access to versions of the category that will not be changed.

  Since categories are both the granule of release and reuse, it stands to reason that the dependencies that we wish to manage are the dependencies *between* categories rather than the dependencies *within* categories. After all, within a category, classes are expected to be highly interdependent. Since all the classes within a category are reused at the same time, and since all classes in a category are closed against the same kind of changes, the interdependence between them cannot do much harm.

Starting from these statements, Martin provides the following metric definitions:

- **Ca - Afferent Couplings:** The number of classes outside this category that depend upon classes within this category;
- **Ce - Efferent Couplings:** The number of classes inside this category that depend upon classes outside this categories;
- **I - Instability:**

  $$\frac{Ce}{Ca + Ce} \tag{A.1}$$

  This metric has the range [0,1]. I=0 indicates a maximally stable category. I=1 indicates a maximally instable category.

In this study only Ce metric has been computed and only for the analyzed Java projects. The computation has been automatically performed by the TEAMINABOX Eclipse Metrics Plugin. Where the dependence has been checked in terms of inheritance, interface implementation, parameter types, variable types, thrown and caught exceptions. In short, all types referred to anywhere within the source of the measured class. Most important is that the the concept of category has been lost in the computation, thus the category becomes the same as the class itself.

- **LCOM-CK - CK's Lack of Cohesion in Methods** - Gives a measure of the cohesion between the methods of a class. The formal definition has been given in equation 1.7. This metric has been computed only on the Java systems using the TEAMINABOX Eclipse Metrics Plugin. This metric is part of the Chidamber and Kemerer suite.

- **LCOM-HS - HS's Lack of Cohesion in Methods** - Gives a measure of the cohesion between the methods of a class. This metric has been proposed by Henderson-Sellers as an alternative definition for the LCOM metric preposed by Chidamber and Kemerer. The major critique to the original version is that with the LCOM-CK, there are a lot of dissimilar examples all giving a zero value. Hence, while a large value of LCOM suggests a poor cohesion, the zero value does not necessarily indicate good cohesion. More over the original definition lacked of guidelines for the measure interpretation [Henderson-Sellers et al., 1996].

Thus, he suggested that the requirements for LCOM definition must include:

1. The ability to give values across the full range;
2. The measure must give values which can be uniquely interpreted in terms of cohesion.

Henderson-Sellers defines Lack of Cohesion in Methods as follows.
Let:

M be the set of methods defined by the class (see note 1 below);

F be the set of fields defined by the class (see note 2 below);

$r(f)$ be the number of methods that access field f, where f is a member of F;

$< r >$ be the mean of $r(f)$ over F.

Then:

$$LCOM = \frac{< r > -|M|}{1 - |M|} \tag{A.2}$$

Note 1: The TEAMINABOX Metric Plugin only includes methods in M if they access at least one field. The reason for this is that methods that do not access fields are often required to be non-static for reasons of polymorphic dispatch. However, these kinds of methods skew the value of this metric in a way that is not helpful.

Note 2: The TEAMINABOX Metric Plugin only includes fields in F if they are accessed by at least one method in the class. A good compiler can tell you if fields are unused, and leaving them in the calculation of this metric skews the value.

Appendix B

Statistics

This section collects for each analyzed system the descriptive statistics of all computed metrics. Tables B.1 - B.6 show the descriptive statistic for all the Smalltalk system. Tables B.7 and B.8 show the descriptive statistic respectively for the Ant and for the Tomcat systems. The descriptive statistics are given in terms of Max, Min, Median, Second and Third percentiles, Mean and standard Deviation.

METRIC	*Max*	*Min*	*75^o Percentile*	*Median*	*25^o Percentile*	*Mean*	*Std. Deviation*
InputDegree	$1.3016 \cdot 10^4$	$0.0000 \cdot 10^4$	$0.0057 \cdot 10^4$	$0.0019 \cdot 10^4$	$0.0003 \cdot 10^4$	$0.0067 \cdot 10^4$	$0.0305 \cdot 10^4$
OutputDegree	$2.5900 \cdot 10^3$	$0.0010 \cdot 10^3$	$0.0720 \cdot 10^3$	$0.0240 \cdot 10^3$	$0.0070 \cdot 10^3$	$0.0673 \cdot 10^3$	$0.1363 \cdot 10^3$
LOC	$5.2040 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.1582 \cdot 10^3$	$0.0610 \cdot 10^3$	$0.0210 \cdot 10^3$	$0.1453 \cdot 10^3$	$0.2711 \cdot 10^3$
WMC	356.0000	0.0000	19.0000	8.0000	3.0000	16.2102	24.1195
DIT	12.0000	0.0000	4.0000	3.0000	2.0000	3.4038	1.9357
NOC	367.0000	0.0000	1.0000	0.0000	0.0000	1.0000	8.1438
RFC	902.0000	0.0000	53.2500	23.0000	9.0000	42.0401	58.7881

Table B.1: *Statistics. Statistics for each metric computed on the VisualWorks base image (Graph#1).*

METRIC	*Max*	*Min*	*75^o Percentile*	*Median*	*25^o Percentile*	*Mean*	*Std. Deviation*
InputDegree	$1.4836 \cdot 10^4$	$0.0000 \cdot 10^4$	$0.0061 \cdot 10^4$	$0.0019 \cdot 10^4$	$0.0004 \cdot 10^4$	$0.0070 \cdot 10^4$	$0.0326 \cdot 10^4$
OutputDegree	$2.6640 \cdot 10^4$	$0.0010 \cdot 10^4$	$0.0740 \cdot 10^4$	$0.0250 \cdot 10^4$	$0.0070 \cdot 10^4$	$0.0699 \cdot 10^4$	$0.1412 \cdot 10^4$
LOC	$5.2040 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.0720 \cdot 10^3$	$0.0240 \cdot 10^3$	$0.0070 \cdot 10^3$	$0.1453 \cdot 10^3$	$0.2711 \cdot 10^3$
WMC	356.0000	0.0000	20.0000	9.0000	4.0000	16.7886	25.0157
DIT	12.0000	0.0000	4.0000	3.0000	2.0000	3.3544	1.9035
NOC	419.0000	0.0000	1.0000	0.0000	0.0000	1.0000	8.7231
RFC	902.0000	0.0000	55.0000	24.0000	9.0000	43.4089	61.1317

Table B.2: *Statistics. Statistics for each metric computed on the VisualWorks image with the Store parcels loaded (Graph#2).*

METRIC	*Max*	*Min*	*75^o Percentile*	*Median*	*25^o Percentile*	*Mean*	*Std. Deviation*
InputDegree	$1.0067 \cdot 10^4$	$0.0000 \cdot 10^3$	$0.0059 \cdot 10^4$	$0.0021 \cdot 10^4$	$0.0005 \cdot 10^4$	$0.0067 \cdot 10^4$	$0.0280 \cdot 10^4$
OutputDegree	$1.4110 \cdot 10^3$	$0.0010 \cdot 10^3$	$0.0760 \cdot 10^3$	$0.0290 \cdot 10^3$	$0.0097 \cdot 10^3$	$0.0674 \cdot 10^3$	$0.1215 \cdot 10^3$
LOC	$1.1174 \cdot 10^6$	$0.0000 \cdot 10^6$	$0.0421 \cdot 10^6$	$0.0140 \cdot 10^6$	$0.0046 \cdot 10^6$	$0.0404 \cdot 10^6$	$0.0796 \cdot 10^6$
WMC	234.0000	0.0000	20.0000	9.0000	4.0000	17.0944	23.9047
DIT	12.0000	0.00000	4.0000	3.0000	2.0000	3.4475	2.0427
NOC	264.0000	0.0000	1.0000	0.0000	0.0000	1.0000	6.8157
RFC	550.0000	0.0000	55.0000	26.0000	11.0000	43.4890	55.5285

Table B.3: *Statistics. Statistics for each metric computed on the short version of the VisualWorks image (Graph#3).*

METRIC	*Max*	*Min*	*75^o Percentile*	*Median*	*25^o Percentile*	*Mean*	*Std. Deviation*
InputDegree	543.7770	0.0000	76.0730	21.4701	6.7333	57.9806	83.3947
OutputDegree	$1.4390 \cdot 10^3$	$0.0010 \cdot 10^3$	$0.0860 \cdot 10^3$	$0.0370 \cdot 10^3$	$0.0100 \cdot 10^3$	$0.0931 \cdot 10^3$	$0.1800 \cdot 10^3$
LOC	$2.1510 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.2400 \cdot 10^3$	$0.0900 \cdot 10^3$	$0.0380 \cdot 10^3$	$0.2075 \cdot 10^3$	$0.3170 \cdot 10^3$
WMC	270.0000	0.0000	27.0000	11.0000	5.0000	22.4333	31.7312
DIT	7.0000	1.0000	4.0000	3.0000	2.0000	3.0905	1.5732
NOC	7.0000	0.0000	1.0000	0.0000	0.0000	0.6190	1.2932
RFC	615.0000	0.0000	66.0000	31.0000	13.0000	56.6571	79.5737

Table B.4: *Statistics. Statistics for each metric computed on VisualWorks Store parcels (Graph#4).*

METRIC	*Max*	*Min*	*75^{o}Percentile*	*Median*	*25^{o}Percentile*	*Mean*	*Std. Deviation*
InputDegree	$1.4836 \cdot 10^4$	$0.0000 \cdot 10^4$	$0.0247 \cdot 10^4$	$0.0057 \cdot 10^4$	$0.0008 \cdot 10^4$	$0.0302 \cdot 10^4$	$0.1138 \cdot 10^4$
OutputDegree	629.0000	87.0000	1.0000	24.5000	3.0000	64.7043	101.9011
LOC	$1.7280 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.2100 \cdot 10^3$	$0.0695 \cdot 10^3$	$0.0140 \cdot 10^3$	$0.1642 \cdot 10^3$	$0.2425 \cdot 10^3$
WMC	249.0000	0.0000	26.0000	9.0000	2.0000	20.4247	30.5548
DIT	7.0000	0.0000	5.0000	3.0000	2.0000	3.3925	1.7620
NOC	419.0000	0.0000	2.0000	0.0000	0.00000	3.8441	30.9399
RFC	364.0000	0.0000	55.0000	22.0000	4.0000	39.5054	53.1350

Table B.5: *Statistics. Statistics for each metric computed on the VisualWorks Core classes (Graph#5).*

METRIC	*Max*	*Min*	*75^o Percentile*	*Median*	*25^o Percentile*	*Mean*	*Std. Deviation*
InputDegree:	$8.9831 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.0766 \cdot 10^3$	$0.0235 \cdot 10^3$	$0.0051 \cdot 10^3$	$0.1012 \cdot 10^3$	$0.3961 \cdot 10^3$
OutputDegree	$5.1468 \cdot 10^3$	$0.0010 \cdot 10^3$	$0.0980 \cdot 10^3$	$0.0340 \cdot 10^3$	$0.0080 \cdot 10^3$	$0.1012 \cdot 10^3$	0.2471
LOC	$7.3770 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.1770 \cdot 10^3$	$0.0640 \cdot 10^3$	$0.0220 \cdot 10^3$	$0.1840 \cdot 10^3$	$0.4244 \cdot 10^3$
WMC	$1.0900 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.0220 \cdot 10^3$	$0.0100 \cdot 10^3$	$0.0040 \cdot 10^3$	$0.0202 \cdot 10^3$	$0.0429 \cdot 10^3$
DIT	10.0000	0.0000	5.0000	4.0000	3.0000	3.8682	1.6347
NOC	410.0000	0.0000	0.0000	0.0000	0.0000	1.0000	10.0404
RFC	$1.7930 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.0670 \cdot 10^3$	$0.0300 \cdot 10^3$	$0.0110 \cdot 10^3$	$0.0553 \cdot 10^3$	$0.0903 \cdot 10^3$

Table B.6: *Statistics. Statistics for each metric computed on the Squeak base image (Graph#6).*

METRIC	*Max*	*Min*	*75^{o}Percentile*	*Median*	*25^{o}Percentile*	*Mean*	*Std. Deviation*
outputDegree	149.0000	0.0000	7.0000	3.0000	1.0000	5.8433	9.8184
inputDegree	152.0000	0.0000	1.0000	1.0000	0.0000	2.1456	8.6221
LOC	518.0000	56.0000	71.0000	64.0000	61.0000	69.3856	29.4673
Ce	54.0000	2.0000	17.0000	10.0000	7.0000	12.9068	8.4778
LCOM-CK	$1.2990 \cdot 10^4$	$0.0000 \cdot 10^4$	$0.0075 \cdot 10^4$	$0.0000 \cdot 10^4$	$0.0000 \cdot 10^4$	$0.0281 \cdot 10^4$	$0.1186 \cdot 10^4$
LCOM-HS	100.0000	0.0000	86.0000	73.0000	0.0000	55.9047	36.9537
WMC	279.0000	2.0000	41.0000	20.0000	10.0000	32.5614	36.8838

Table B.7: *Statistics. Statistics for each metric computed for Ant system.*

METRIC	*Max*	*Min*	*75º Percentile*	*Median*	*25º Percentile*	*Mean*	*Std. Deviation*
outputDegree	113.0000	0.0000	15.0000	6.0000	2.0000	11.8182	17.4705
inputDegree	115.0000	0.0000	2.0000	0.0000	0.0000	2.7810	9.9783
LOC	714.0000	61.0000	88.0000	77.0000	71.0000	90.5950	71.8778
Ce	76.0000	2.0000	23.0000	12.0000	6.0000	15.8223	12.8476
LCOM-CK	$4.5820 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.0040 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.0000 \cdot 10^3$	$0.0435 \cdot 10^3$	$0.3045 \cdot 10^3$
LCOM-HS	100.0000	0.0000	83.0000	50.0000	0.0000	45.1818	38.0667
WMC	434.0000	2.0000	60.0000	27.0000	11.0000	50.6033	62.6515

Table B.8: *Statistics. Statistics for each metric computed for Tomcat system.*

Bibliography

[Abreu and Carapuca, 1994] Abreu, F.B., Carapuca, R., *Object-oriented software engineering: Measuring and controlling the development process*, Proceeding of the 4th International Conference on Software Quality, 1994.

[Abreu et al., 1995] Abreu, F.B., Goulao, M., Esteves, R., *Toward the Design Quality Evaluation of Object-Oriented Software Systems*, Proceeding of the 5th International Conference on Software Quality, 1995.

[Aiello et al., 2000] Aiello, W., Chung, F., Lu, L., *A random graph model for massive graphs*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Association of Computing Machinery, New York, pp. 171180, 2000

[Albert et al., 1999] Albert, R., Jeong, H., Barabasi, A., *Diameter of the World Wide Web*, Nature Vol. 401, pp 130-131, 1999

[Alshayeb and Li, 2003] Alshayeb, M., Li, W., *An Empirical Validation of Object Oriented Metrics in two Different Iterative Software Processes*, IEEE Transaction on Software Engineering, 29(11) pp. 1043-1049, 2003

[Amaral et al., 2000] Amaral, L.A.N., Scala, A., Barthèlèmy, M., Stanley, H.E., *Classes of small-world networks*, Proc. Natl. Acad. Sci. U.S.A. 97 (21), pp. 11149-11152, 2000

[Basili and Rombach., 1988] Basili, V.R., Rombach, H.D., *The TAME Project: Towards Improvement-Oriented Software Environment*, IEEE Transaction on Software Engineering, 14(6) pp. 758-73, 1988

[Basili, 1992] Basili, V.R., *Software Modeling and Measurement: The Goal Question Metric Paradigm* Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD, September 1992.

[Basili et al., 1996] Basili, V.R., Briand, L.C., Melo, W.L., *A validation of object-oriented design metrics as quality indicators*, IEEE Transaction on Software Engineering, 22(10) pp. 751-61, 1996

[Barabasi and Albert, 1999] Barabasi, A., Albert, R., *Emergence of scaling in random networks*, Science Vol. 286, pp. 509-512, 1999

[Barabasi et al., 1999] Barabasi, A., Albert, R., Jeong, H., *Scale-free characteristics of random networks: the topology of the World Wide Web* , Phys. A Vol. 281, pp 69-77, 2000

[Bollobás, 1981] Bollobás, B., *The diameter of random graphs*, Trans. Amer. Math. Soc., 267, pp. 4152, 1981

[Bollobás, 1995] Bollobás, B., *Random Graphs*, Academic Press, London, 1985

[Bollobás and Riordan, 2002] Bollobás, B., Riordan, O., *The Diameter of a Scale-Free Random Graph*, preprint, Department of Mathematical Sciences, University of Memphis, 2002.

[Briand et al., 1999] Briand, L.C. , Wst, J., Ikonomowsky, S., Luonis, H., *Investigating Quality Factors in Object-Oriented Designs: an Industrial Case Study*, Proceedings of the 21st International Conference on Software Engineering, 51(3) pp. 245-54, 1999

[Briand et al., 2000] Briand, L.C. , Wst, J., Daly, J.W., Porter, D.V., *Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems*, Journal of Systems and Software, 51(3) pp. 245-73, 2000

[Broder et al., 2000] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J., *Graph structure in the web*, Computer Networks, 33, pp. 309320, 2000

[Brooks, 1987] Brooks, F.P., *No Silver Bullet: Essence and Accidents of Software Engineering*, Computer, 20(4) pp. 10-19, 1987

[Brooks, 1995] Brooks, F.P., *The Mythical Man-Month*, Addison-Wesley, 1995

[Bunge, 1979] Bunge, M., *Treatise on Basic Philosophy: Ontology II: the World of Systems*, Riedel, Boston, MA, 1979

[Chen et al, 2002] Chen, Q., Chang, H., Govindam, R., Jamin, S., Shenker, S.J., *The origin of power laws in Internet topologies revisited*, Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE Computer Society, Los Alamitos, CA, 2002.

[Chidamber and Kemerer, 1994] Chidamber, S.R., Kemerer, C.F., *A Metric Suite for Object Oriented Design*, IEEE Transaction on Software Engineering, 20(6) pp. 476-93, 1994

[Chidamber at al, 1994] Chidamber, S.R., Darcy, D.P., Kemerer, C.F., *Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis*, IEEE Transaction on Software Engineering, 24(8) pp. 629-39, 1994

[Chung and Lu, 2002] Chung, F., Lu, L., *The average distances in random graphs with given expected degrees*, Proc. Natl. Acad. Sci. USA, 99, pp. 1587915882, 2002

[Churcher and Shepperd, 1995] Churcher, N.I., Shepperd, M.J., *Comment on - A Metric Suite for Object Oriented Design*, Correspondence of IEEE Transaction on Software Engineering, 21(3) pp. 263-65, 1995

[Concas et al., 2005] Concas, G., Marchesi, M., Pinna, S., Serra, N., *Power-laws in a Large Object-Oriented Software System*, Submitted for Publication to PNAS, 2005

[Crovella and Taqqu, 1999] Crovella, M.E., Taqqu, M.S., *Estimating the heavy tail index from scaling properties*, Methodology and computing in applied probability 1 (1), pp. 5579, 1999

[De Castro and Grossman, 1999] De Castro,R., Grossman, J.W., *Famous trails to Paul Erdos*, Math. Intelligencer, 21, pp. 5163, 1999

[Erdős and Rényi, 1959] Erdős, P., Rényi, A., *On random graphs*, I, Publ. Math. Debrecen 6, pp. 290-291, 1959

[Erdős and Rényi, 1960] Erdős, P., Rényi, A., *On the Evolution of random graphs*, Publ. Math. Inst. Hungar. Acad. Sci. 5, pp. 17-61, 1961

[Erdős and Rényi, 1961] Erdős, P., Rényi, A., *On the strength of connectedness of a random graphs*, Acta Math. Hungar. Acad. Sci. 12, pp. 261-67, 1961

[Faloutsos, 1999] Faloutsos, M., Faloutsos, P., Faloutsos, C., *On power-law relationships of the Internet topology*, Computer Communications Rev., 29, pp. 251262, 1999

[Fenton, 1997] Fenton, N.E., Pfleeger, S.L., *Software Metrics, a Rigorous and Practical Approach*, Second Edition, PWS Publishing Company, Boston, MA, 1997

[Focardi et al., 2000] Focardi, S., Marchesi, M., Succi, G., *Extreme Programming Examined*, Addison-Wesley, pp. 191-206, 2000

[Frund and Simon, 1997] Frund, J.E., Simon, G.A, *Modern Elementary Statistics*, Prentice-Hall, Upper Saddle River, NJ, 1997

[Grady and Caswell, 1987] Grady, R.B., Caswell, D., *Software Metrics: Establishing a Company-wide Program*, Prentice Hall, Englewood Cliffs, NJ, 1987

[Grossman, 1995] Grossman, J.W., Ion, P.D.F., *On a portion of the well-known collaboration graph*, Congr. Numer., 108, pp. 129131, 1995

[Henderson-Sellers et al., 1996] Henderson-Sellers, B., Constantine, L.L., Graham, I.M.,*Coupling and cohesion: towards a valid metrics suite for object-oriented analysis and design*, Object Oriented Systems, 3(3), pp. 143-58, 1996

[Hill, 1975] Hill, B.M., *A simple general approach to inference about the tail of a distribution*, The Annals of Statistics, 3, pp. 1163-74, 1975

[Janson et al., 1999] Janson, S., Luczak, T., Rucinski, A. *Random Graphs*, John Wiley, New York, 1999

[Janson, 2000] Janson, S., *Growth of components in random graphs*, Random Structures and Algorithms 17, pp. 343-356, 2000

[Jeong et al., 2001] Jeong, H., Mason, S., Barabasi, A., Oltvai, Z.N., *Lethality and centrality in protein networks*, Nature, 411, pp. 4142, 2001

[Li and Henry, 1993] Li, W. and Henry, S., *Object-Oriented Metrics that Predict Maintainability*, Journal of Systems and Software, vol. 23, pp. 111-122, 1993.

[Li and Henry, 1993] i, W. and Henry, S.,, *Maintenance Metrics for the Object-Oriented Paradigm*, Proceedings of the First International Software Metrics Symposium, pp. 52-60, Baltimore, MD, 1993

[Liljeros et al., 1990] Liljeros, F., Edling, C.R., Amaral, L.A.N., Stanley, H.E., Aberg, Y., *The web of human sexual contacts*, Nature, 411, pp. 907908, 2001

[Los Tres Amigos, 1999] Booch, G., Jacobson, I., Rumbaugh, J., *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, 1999

[Luczak, 1990] Luczak, T., *Component behavior near the critical point of the random graph process*, Random Structures and Algorithms 1, pp. 287310, 1990.

[Luczak et al., 1994] Luczak, T., Pittel, B.G., Wierman, J.C., *The structure of a random graph near the point of the phase transition*, Transactions of the American Mathematical Society 341, pp. 721-48, 1994.

[Luczak, 1996] Luczak, T., *The phase transition in a random graph*, In Combinatorics, Paul Erdos is Eighty, vol. 2, pp. 399-422, eds. D. Miklos, V.T. Sos & T. Szonyi, Budapest, 1996.

[Marchesi et al., 2003] Marchesi, M., Pinna, S., Serra, N., Tuveri, S., *Power Laws in Smalltalk*, Twelfth Smalltalk Joint Event, Koethen, Germany, 2004

[Martin, 1994] Martin, R., *OO Design Quality Metrics, An Analysis of Dependencies*, objectmentor.com, 1994

[McCabe, 1976] McCabe, T., *A software complexity measure*, IEEE Transaction on Software Engineering, SE-2(4) pp. 308-20, 1976

[Milgram, 1967] Milgram, S., *The small world problem*, Psych. Today, 2, pp.60,67, 1967

[Myers, 2003] Myers, C.R., *Software systems as complex networks: structure, function, and evolvability of software collaboration graphs*, Phys. Rev. E 68, 046116, 2003

[Newman, 2001] Newman, M.E.J., *Scientific col laboration networks: I. Network construction and fundamental results*, Phys. Rev. E, 64, art. no. 016131, 2001

[Newman, 2001] Newman, M.E.J., *The structure of scientific col laboration networks*, Proc. Natl. Acad. Sci. USA, 98, pp. 404409, 2001

[Parnas, 1972] Parnas, D.L., *On the criteria to be used in decomposing systems into modules*, Communications of the ACM, 15(12), pp 1052-8, 1972

[Potanin et al., 2002] Potanin, A., Noble, J., Frean, M., Biddle, R., *Scale-free geometry in Object Oriented programs*, Victoria University of Wellington, New Zeland, Technical Report CS-TR-02/30, 2002

[Render, 1998] Render, S., *How popular is your paper: an empirical study of the citation distribution*, Eur. Phys. J.B., 4, pp. 131-34, 1998

[Saucier, 2000] Saucier, R., *Computer Generation of Statistical Distributions*, Army Research Laboratory, ARL-TR-2168, 2000

[Serra, 2001] Serra, N., *Rappresentazione di un Sistema Software e Simulazione del Processo di Manutenzione Utilizzando i Grafi Casuali*, Tesi di Laurea, DIEE, Università di Cagliari, 2001

[Solomonoff and Rapport, 1951] Solomonoff, R., Rapport, A., *Connectivity of random nets*, Bull. Math. Biophys. 13, pp. 107-17, 1951

[Subramanyam and Krishnan, 2003] Subramanyam, R., Krishnan, M.S., *Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects*, IEEE Transaction on Software Engineering, 29(4) pp. 297-310, 2003

[Tuveri, 2004] Tuveri, S., *Analisi dell'Evoluzione di Sistemi Open Source Utilizzando Modelli Basati sui Grafi Casuali*, Tesi di Laurea, DIEE, Università di Cagliari, 2004

[Valverde et al., 2002] Valverde, S., Cancho, R.F., Solé, R.V., *Scale-Free networks from optimal design*, Eur. Phys. Letters 60, pp. 512-517, 2002

[Valverde and Solé, 2003] Valverde, S., Solé, R.V., *Hierarchical small worlds in Software Architecture*, Submitted to IEEE Transactions of Software Engineering, 2003

[Vázquez et al., 2002] Vázquez, A., Pastor-Satorras, R., Vespignani, A., *Large-scale topological and dynamical properties of the Internet*, Phys. Rev. E, 65, art. no. 066130, 2002

[Wand and Weber, 1990] Wand, Y., Weber, R., *An Ontological Model of an Information System*, IEEE Transaction on Software Engineering, 16(11) pp. 1282-92, 1990

[Watts and Strogatz, 1998] Watts, D.J. Strogatz, S.H., *Collective dynamics of 'small-world' networks*, Nature Vol. 393, pp. 440-442, 1998

[Wheeldon and Counsell, 2003] Wheeldon, R., Counsell, S., *Power law distributions in class relationships*, Proc. Third IEEE Int. Workshop on Source Code Analysis & Manipulation, Amsterdam, The Netherland, 2003

[Wonnacott, 1998] Whonnacott, T.H., Wonnacott, R.J, *Introduzione alla statistica*, Franco Angeli Editore, Milano, It, 1998, italian translation, Vitali, O., "Introductory Statistic", John Wiley & Sons, New York, USA