A Qualitative Method for Measuring the Structural Complexity of Software Systems Based on Complex Networks

Yutao Ma, Keqing He, and Dehui Du
State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, P.R. China
Complex Networks Research Center, Wuhan University, Wuhan 430072, P.R. China
mmyytt_2000@etang.com, hekeqing@public.wh.hb.cn, d_dehui@hotmail.com

Abstract

How can we effectively measure the complexity of a modern complex software system has been a challenge for software engineers. Complex networks as a branch of Complexity Science are recently studied across many fields of science, and many large-scale software systems are proved to represent an important class of artificial complex networks. So, we introduce the relevant theories and methods of complex networks to analyze the topological/structural complexity of software systems, which is the key to measuring software complexity. Primarily, basic concepts, operational definitions, and measurement units of all parameters involved are presented respectively. Then, we propose a qualitative measure based on the structure entropy that measures the amount of uncertainty of the structural information, and on the linking weight that measures the influences of interactions or relationships between components of software systems on their overall topologies/structures. Eventually, some examples are used to demonstrate the feasibility and effectiveness of our method.

1. Introduction

In order to achieve intellectual control over the enormous complexity of a sophisticated software system, measures of software complexity have been a great challenge for software researchers. Sound metrics for the structural complexity of software systems are the key to rationally measuring software complexity, which has always been an urgent demand for software engineering. Although many metrics, measures, and models have been put forward so far, in general, the above problem hasn't been effectively solved yet. P. F. Brooks considers that the theory that gives us a metric for the information embedded in structure (viz. the structural information) is a great challenge for computer science [1]. For example, we know that a Windows XP system is a more complex structure than a Windows 98 system, but how to represent and measure the structural complexity of a software system is not a simple problem.

Complex networks are recently studied across many fields of science, and many of them have been shown to share global statistical features such as the "small world" property [2] of short paths between any two nodes and highly clustered connections and "scale free" property [3] (if a node has k edges, the degree distribution $p(k)$ decays as a power law $p(k) \sim k^{-r}$, where r is often between 2 and 3). Large-scale software systems represent an important class of artificial complex networks, and some also have been found to share "small world" and "scale free" properties [4,5]. These features reveal and characterize the topological or structural information of complex systems, which provides the foundation for our researches. So, the purpose of this paper is to propose a qualitative method to measure the structural complexity of software systems based on complex networks, which can simplify the complexity of software structure so as to provide the underlying basis for the researches on software complexity.

This paper is arranged as follows. Section 2 introduces the related work. Section 3 introduces a directed graphic representation for software structure and presents all parameters of the method. Section 4 introduces details of the method and discusses results of the pre-designed experiments. In the end, section 5 summarizes and concludes this paper.

2. Related work

Software complexity metrics have been deemed as a significant research field of software engineering. From the 1960s to the 1970s, there were three popular

measures of structured programming: Line of codes (LOC) based estimation, Halstead's Software Science metrics (HSS) [6], and McCabe's Cyclomatic Complexity measure (MCC) [7]. In the early 1980s, a few new metrics for modular programming that took interactions between modules or subroutines into consideration emerged, and the famous measures included: Software Structure Metrics based on Information Flow (HIF) [8], Emerson's metric for module cohesion (MC) [9], Stability Measures proposed by Yau and Collofello (YC) [10], and so on. Object-Oriented (OO) programming has been a prominent technique since the early 1990s, so OO software metrics have been widely applied to practical project development. The most-used metrics include: Chidamber & Kemerer object-oriented metrics (CK6) [11], Lorenz & Kidd object-oriented software metrics (LK) [12], Metrics for Object-Oriented Design (MOOD) set [13], etc. However, these metrics or measures are hard to describe the structural complexity of modern complicated software systems in the round.

In both the organic forms of nature and the engineered artifacts of human society, complex systems grow and evolve to reveal intricately networked organizations. Surprisingly, the underlying structures of these networks such as the Internet and the World Wide Web (WWW) have recently been found to share many "scale free" and "small world" qualities, which can be rather different from those found in simple random networks. These discoveries have served to draw together many disparate fields into an emerging science of "complex networks", which aims to uncover the principles by which networked systems form, evolve, and remain robust and adaptable in the face of changing environments [14]. Software systems represent another important class of complex networks, and they to date have received relatively little attention in this field. Although some software systems have been found to share "scale-free" and "small-world" properties, these researches [4,5,15] are performed from the viewpoint of statistical physics, information theory, and Complexity Science, rather than software engineering. Hence, how can we analyze the structural complexity of software systems based on the relevant principles and methods of complex networks so as to form a suit of new metrics (viz. the combination of software engineering and Complexity Science) is a significant problem to be solved in future.

3. Representation and parameters

3.1. Directed graphic representation

Software is built up out of many interacting units and subsystems (or generally called components) at many levels of granularity (subroutines, classes, source files, libraries, etc.), and interactions and collaborations of those pieces can be used to define networks or graphs that form a skeletal description of a software system. If we use a vertex and the directed edge that links two different vertexes to denote a component and the interaction or relationship between components respectively, the structure/topology of a software system can be described as a directed graph regardless of the actual application context.

There are three kinds of graphs used to describe the structure of a software system in practical development processes: (1) *Call graphs* that describe the calling of subroutines or methods by one another have long been used to understand the structure and execution of procedural activity in software systems; (2) *class and/or object collaboration diagrams* are used to glean insight into the relationships or interactions among abstract data types in OO systems; and (3) *Component and Connector (C&C) plots* [16] are often used to depict the overall topology or structure of a software system from the viewpoint of software architecture. In Tab.1, UML diagrams are created with Poseidon for UML (a free version). Every module, class, object, or software component maps to a single graph node, and relationships or interactions transform into directed links. There is no semantic distinction among call, inheritance, association, and connector in a directed graph that describes the structure of a system.

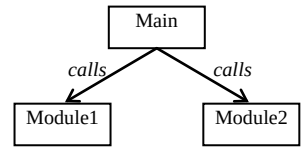
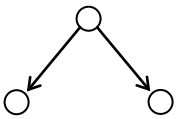
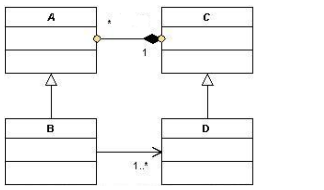
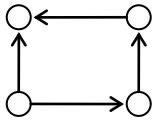
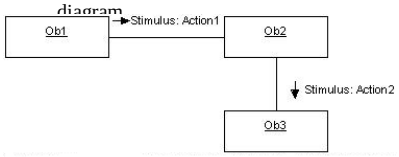
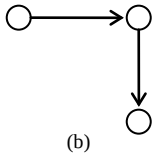
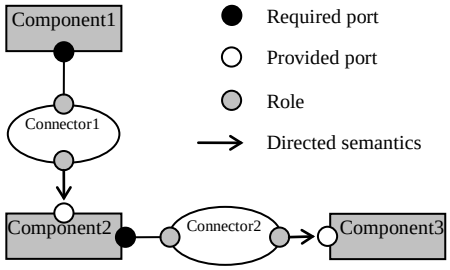
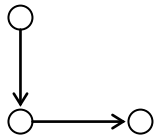
3.2. Parameters of the method

Connectivity. For each node *i* in a directed graph (in this context, we do not make any difference between the terms graph and network), there is both an in-degree (viz. the number of incoming edges to node *i*) and an out-degree (viz. the number of outgoing edges from node *i*). So, the degree of node *i* is the sum of its in-degree and out-degree. Degree distributions, summarizing the connectivity of each node in a graph, are a feature that distinguishes many complex networks from simple random graphs. The directed edge that connects a node to itself represents the relationship or interaction between components hidden in the node (a component), and the degree of the node is zero.

The degree of a node is a special feature that reveals its significance in complex networks, but this parameter is hard to describe the influences on other nodes that indirectly connect to the node, and on the whole graph. Hence, we introduce a new parameter—ripple degree.

Ripple degree. For each node in a directed graph, we define the reachability set $S(v)$ as the set of nodes reachable from v (a node w is reachable from v if there is a directed path from v to w). This is the set of nodes reached by standard depth-first search started at node v. So, the ripple degree of a node is the size of its reachability set (taking into account that every node is self-reachable), which determines a node's influences on other nodes and the whole graph. Furthermore, it can be easily computed in that there are many effective methods of finding the transitive closure of an adjacent matrix that represents a directed graph.

Table 1. **A directed graphic representation for software structure**

Category	Traditional representation	Directed Graph
Call graph		
Collaboration diagram	 (a) A simple UML class diagram	 (a)
	 (b) A simple UML object collaboration diagram	 (b)
C&C plot		

Recently, meta-modeling as a pervasive approach of meta-computing has received widespread attention, and the architecture of a meta-model such as OMG's MOF (Meta-Object Facility) [17] and UML meta-model is being widely applied to modeling and designing software systems. So, the significance of a component or unit of software systems is also related to its type and abstract level that can be described by the abstraction degree of a node (component).

Abstraction degree. The abstraction degree of a node ($a(v)$) is defined as the abstract level of its corresponding component in the meta-modeling architecture, and it is given by:

$$a(v) = MN + 1 \quad (1)$$

where MN is the number of meta or abstraction. So, if v denotes a component that is an object or instance, $a(v)$ equals 1. This parameter used to describe the vertical order of a node presents a striking contrast to the above parameter that expresses the horizontal influence of a node.

The complexity of a system depends not only on the number of its components, but also on the interactions between its components. So, the number and type of interactions or relationships are also significant factors that determine the complexity of software structure.

Weight. We define the weight of an edge ($w(e)$) as the significance of its corresponding interaction or relationship in software systems. It is easy to translate relationships in call graph and OO systems (such as inheritance, composite, and collaboration) to different numeric weights. For each system built up with software components under the principles of Component-Based Software Development (CBSD), in Tab.2, we show different weights of basic connectors

according to a connector taxonomy defined in [18]. So, the weight of an edge is:

$$w(e): f(w_i, \theta) \rightarrow \mathbf{R} \quad (2)$$

where $w_i \in$ (Weight $\rightarrow$ **R**) and θ is the threshold chosen by designers. Furthermore, a connector may be a composite one that contains other connectors, and the weight is defined as:

$$w(e): f(\sum_{i=1}^{n} w_{ji} x_i, \theta_j) \rightarrow \mathbf{R} \quad (3)$$

where w_{ji} is the weight of x_i (the component of a composite connector) and θ_j is the threshold of x_i (assuming that there is a linear relation between w_j and w_{ji}). When weights of all edges have been taken into account, a software system's structure can be depicted by a weighted directed graph.

Table 2. **Weights of basic software connectors**

Service	Type	Weight
Facilitation	Linkage	A
Communication	Stream	B^-
Communication Conversion	Data Access	B^+
Communication Coordination	Procedure Call	C
Communication Coordination	Event	C
Conversion	Adaptor	D
Facilitation	Distributor	E^-
Coordination Facilitation	Arbitrator	E^+

4. Method and experiment

4.1. Method based on structure entropy

In physics, the word entropy has important physical implications as the amount of "disorder" of a system. Entropy as defined by Shannon [19] is closely related to entropy as defined by physicists, and Shannon's information entropy is appropriate for measuring uncertainty over an unordered space for determination of the degree of uncertainty that would preside over expectations, either adaptive or rational.

If a system with n different units has well-defined border and specific functions and topology, let Q (total metamorphic energy of a system's structure) be a discrete random variable taking a finite number of possible values $q_1, q_2, \ldots, q_n$ (the metamorphic energy of a unit's structure) with probabilities $p_1, p_2, \ldots, p_n$ respectively such that:

$$\sum_{i=1}^{n} p_i = 1 \; (\; p_i \geq 0, i = 1, 2, \ldots, n) \quad (4)$$

We attempt to obtain a number that will measure the amount of uncertainty, so let Q be the sum of q_i. Then, p_i (p_i is interpreted as the uncertainty associated with the event) is given by:

$$p_i = q_i \Big/ Q \quad (5)$$

So, the structure entropy can be defined as:

$$E = -\sum_{i=1}^{n} p_i \ln p_i \quad (6)$$

which measures how homogenous a set of units is. If units of a system are approximately homogenous, the structure entropy is large (when all units are completely homogenous, the maximum equals ln$n$, where n is the number of units). Contrarily, if units are very different or heterogeneous, the structure entropy is small.

To measure the structural complexity related to uncertainty of the structural information, we introduce a new parameter:

$$r = 1 - E/E_{\max} \quad (7)$$

If units of a system are homogenous, this implies little uncertainty of the system's structure, so r is small (the structure is simple); whereas, if units are very heterogeneous, r is large, which implies much uncertainty of the system's strcuture. Connectivity, ripple degree, and abstraction degree of a node will influence the overall structure of a network, so we define the evaluation coefficient as:

$$R = \sum_{i=1}^{3} \varphi_i r_i \quad (8)$$

where r_i may be the computing result of connectivity, ripple degree, or abstraction degree according to the formula (7), and φ_i is the computing weight of r_i such that:

$$\sum \varphi_i = 1 \quad (9)$$

Therefore, R can determine the structural complexity of a software system based on the evaluation coefficient.

If the selected systems are too similar to be distinguished by R, we need to take the weights of edges into account. Then, the parameter—weight ratio is defined as:

$$W = (w_{\max} - w_{\min}) * \frac{\sum_{i=1}^{n} w_i}{n} \quad (10)$$

which describes the influences of edges (the type and number) on the structural complexity of a system. The larger W is, the more complex a system's structure will be. So, R and W are both used to qualitatively measure the structural complexity of a software system.

4.2. Experiment and results

In Tab.3, we list evaluation coefficients of different systems that are computed according to the method introduced in section 4.1 to show their structural complexity. The basic data of topological measurements and exponents of cumulative degree distributions are cited from [5]. r_1, r_2, and r_3 denotes the computing result of connectivity, ripple degree, and abstraction degree according to the formula (7) respectively. Because R is able to differentiate listed systems that are intentionally selected by the author, W is omitted.

Lattice-like network is a kind of regular networks, and its structure is simple because all nodes are completely homogenous (R equals 0). Subsystem is a kind of stochastic networks in nature, and its structure is still simple (R is small). The systems from Aime to Linux are all found to share "scale free" property, because of preferential attachment [3] (such as reuse), optimization [20], and copy [4] (such as instantiation), their structures are obviously more complex (R is large). Star network is a special network that only has a center node, which defines another extreme within the universe of complex networks. When the structure entropy gradually decreases (R increases by degrees), uncertainty of the structural information increases and at the same time the structural complexity enhances, which accords with the conclusion achieved through the information entropy proposed in [15].

Table 3. **Results of the qualitative analysis**

System	Node	Edge	r_1	r_2	r_3	$R^{\blacklozenge}$	Degree$^{\bullet}$	In-degree$^{\bullet}$	Out-degree$^{\bullet}$	d	C
Lattice-like*	-	-	0	0	0	0	-	-	-	-	-
Subsystem	25	25	0.027	0.286	0.155	0.156	-	-	-	4.96	0.004
Aime	143	319	0.198	0.472	0.046	0.277	1.43±0.05	1.3±0.04	1.48±0.07	2.66	0.413
Yahoopops	373	711	0.281	0.507	0.014	0.318	1.67±0.05	1.46±0.06	1.69±0.05	5.57	0.336
Blender	495	834	0.276	0.524	0.009	0.323	1.64±0.04	1.36±0.05	2.04±0.09	6.54	0.155
GTK	748	1147	0.236	0.603	≈0	0.336	1.51±0.04	1.22±0.02	2.38±0.2	5.87	0.081
Xmms	971	1802	0.272	0.668	≈0	0.384	1.62±0.03	1.48±0.03	1.86±0.05	6.34	0.084
JDK-B	1364	1947	0.243	0.722	0.012	0.386	1.55±0.8	1.39±0.05	2.3±0.14	5.97	0.225
CS	1488	3526	0.25	0.736	0.006	0.394	1.58±0.02	1.22±0.03	1.96±0.09	3.92	0.135
Striker	2356	6748	0.29	0.812	0	0.441	1.7±0.04	1.54±0.03	1.73±0.06	5.90	0.282
Linux	5285	11359	0.277	0.863	0	0.456	1.65±0.06	1.42±0.02	1.82±0.04	4.66	0.139
Star*	-	-	0.5	1	0	0.6	-	-	-	-	-

$*$: Number of nodes $n \to \infty$; $\blacklozenge$: $R = 0.4r_1 + 0.4r_2 + 0.2r_3$; $\bullet$: $P(k) \sim k^{-r}$, $P_{>}(k) = \sum_{k'>k} P(k') \sim k^{-r+1}$.

Then, we will investigate what influences the structural complexity of software systems that have been found to share "scale free" property and the influences on some basic characteristics of complex networks (such as the average shortest path d and the average clustering coefficient C) when the topology of a software system changes.

Figure 1a shows the relationship between the exponent of cumulative degree distribution (r) and r_1. When r becomes larger, $p(k)$ decays quickly and degrees of nodes distribute more unevenly, so the entropy of connectivity decreases and r_1 tends to increase. Figure 1b shows the relationship between the size of a system (N) and r_2. When N increases with order of magnitudes (such as five times), influences of central nodes will enhance, so the entropy of ripple degree reduces and r_2 tends to become larger. For similar-scaled systems, we find that there is a positive correlation between the exponent of cumulative out-degree distribution (r_{out}) and r_2 (in Figure 1c), but the conclusion demands farther experiments to prove its universality. Figure 1d reveals the relationship between N and C. When N becomes larger, C tends to gradually decreases. Furthermore, if selected systems' sizes are similar, C will become smaller along with the increase of r (in Figure 1e), which indicates that the modularity of a system is close related to N and r. Figure 1f reveals the relationship between r and d. When r enlarges, d tends to increase too. So, we can control d of a system through regulating r to achieve the fast information transmission at low cost.

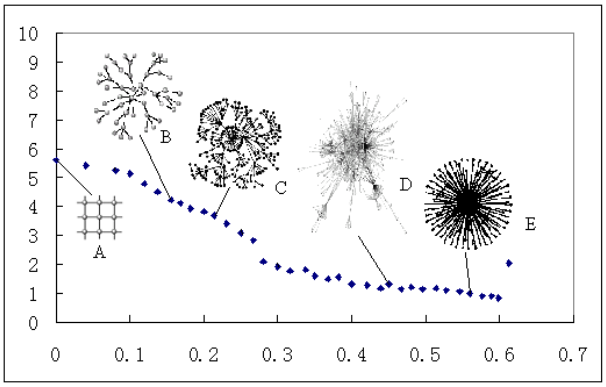


Figure 2. **Relationship between the evaluation coefficient and the structure entropy**

Figure 2 shows the relationship between the evaluation coefficient and the structure entropy on the basis of the above discussions. In Figure 2, A is a lattice-like network; B is an exponential-like network; C is a scale-free network where hubs involving multiple connections and a dominance of nodes with one connection can be seen; E is a star network, and D

is a intermediate graph between C and E in which many hubs can be identified separates sparse nets from dense nets. *When the evaluation coefficient becomes larger, the structure entropy tends to decrease and arrives at the minimum when a system evolves to a star network. Furthermore, uncertainty of the structural information increases, and the structural complexity enhances; at the same time, the robustness of the system is improved because of its more heterogeneous topology, the modularity becomes smaller (C decreases), and the information transmission slows down (d increases).* This is an effective conclusion that can be used to qualitatively analyze the structural complexity of different software systems, which also demonstrates the feasibility and effectiveness of our method.

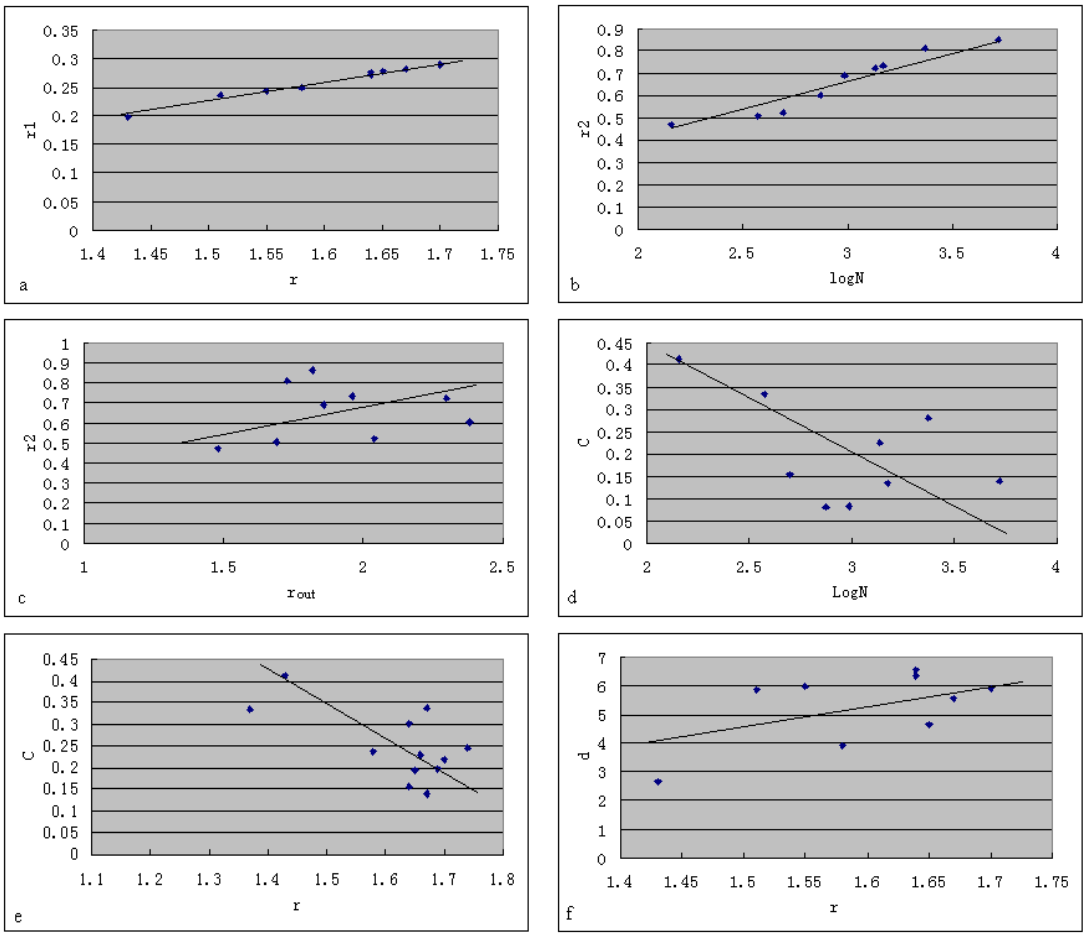


Figure 1. **Analysis on relationships among different factors**

5. Conclusions

Complex networks are recently studied across many fields of science. Many software systems studied exhibit scale-free and/or heavy-tailed degree distributions qualitatively similar to those observed in biological and technological networks, so they represent another important class of complex networks. Complexity metrics for software structure are the key to measuring software complexity that has been a challenge for software engineer for decades. In this paper, we propose a qualitative method to measure the structural complexity of software systems, which combines challenges in software engineering and principles of complex networks and Complexity Science, inspired by some interesting phenomena and questions in complex networks. First of all, we introduce the basic concepts, operational definitions, and measurement units of all parameters that are related to the method. Then, we define the structure entropy that measures the amount of uncertainty of the structural information according to Shannon's information entropy. Furthermore, the evaluation coefficient based on the structure entropy, and the weight ratio measuring influences of the interactions or relationships between components of a software system on its overall topology are both used to qualitatively measure the structural complexity of a software system.

Software systems present novel perspectives to the study of complex networks. The work presented here highlights the need to preserve edge directions in studies of directed software graphs, a fact that has long been recognized within the software engineering community. The "scale free" nature of many software systems reflects these sorts of trade-offs (design patterns, polymorphism, reuse, and related techniques aiming to minimize specificity of interactions while still enabling specific control) in the large. More work is needed to (1) quantitatively measure the structural complexity of a software system (the mapping from our method to OO software metrics such as CK6 and MOOD or to the metrics for structured programming such as fan-in and fan-out) and (2) better abstract and characterize the software development process to uncover the implications of that process for large-scale network topology (the evolutionary mechanism and the control strategy to optimize software development).

Acknowledgement

This work is support by the National Natural Science Foundation of China under grant No.60373086, the Provincial Natural Science Foundation of Hubei Province of China under grant No. 2005ABA123 and No. 2005ABA240, and the Key Scientific Research Project of Education Department of Hubei Province of China under grant No.Z200511005. We gratefully thank anonymous reviewers for their invaluable comments!

References

[1] F.P. Brooks, "Three Great Challenges for Half-Century-Old Computer Science", *Journal of the ACM*, Vol.50, No.1, 2003, pp. 25-26.

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

[3] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks", *Science*, 286, 1999, pp. 509-512.

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

[5] S. Valverde and R.V. Solé, "Hierarchical Small Worlds in Software Architecture", *Working paper of Santa Fe Institute*, SFI/03-07-44, 2003.

[6] M.H. Halstead, *Elements of Software Science*, New York: Elsevier North Holland, 1977.

[7] T.J. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, Vol.2, No.4, 1976, pp. 308-320.

[8] S.M. Henry and D. Kafura, "Software Structure Metrics Based on Information Flow", *IEEE Transactions on Software Engineering*, Vol. SE-7, 1981, pp. 510-518.

[9] T.J. Emerson, "A Discriminate Metric for Module Comprehension", *Proceedings of 7th International Conference on SW-Engineering*, Orlando, Florida, 1984, pp. 294-431.

[10] S.S.Yau and J.S. Collofello, "Some Stability Measures for Software Maintenance", *IEEE Transactions on Software Engineering*, Volume SE-6, 1980, pp. 545-552.

[11] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object-Oriented Design", *IEEE Transactions on Software Engineering*, Vol. 20, 1994, pp. 476-493.

[12] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics, A Practical Guide*, Englewood Cliffs, N.J.: PTR Prentice-Hall, 1994.

[13] F. Brito e Abreu, "The MOOD Metrics Set", *Proceedings of ECOOP'95 Workshop on Metrics*, Aarhus, Denmark, LNCS 952,1995.

[14] A.-L. Barabási, *Linked: The New Science of Networks*, Perseus Publishing, Cambridge, MA, 2002.

[15] R.V. Solé and S. Valverde. "Information Theory of Complex Networks: On Evolution and Architectural Constraints", *Proceedings of International Conference on Complex Networks*, Springer-Verlag, Lect. Notes Phys. 650, 2004, pp. 189-207.

[16] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, New York: Prentice-Hall International, Inc., 1996.

[17] MOF specification, version 1.4, available at WWW: http://www.omg.org/technology/documents/formal/mof.htm.

[18] N.R. Mehta, N. Medvidovic, and S. Phadke, "Towards a taxonomy of software connectors", *Proceedings of the 22nd International Conference on Software Engineering*, Limerick, Ireland, 2000, pp. 178-187.

[19] C.E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, Urbana, Ill, 1949.

[20] S. Valverde, R. Ferrer and R.V. Solé, "Scale Free Networks from Optimal Design", *Europhys. Lett.*, 60, 2002, pp. 512-517.