EUROPHYSICS LETTERS 15 December 2006

Europhys. Lett., **76** (6), pp. 1221–1227 (2006)
DOI: 10.1209/epl/i2006-10384-1

Fractal dimension in software networks

G. Concas, M. F. Locci, M. Marchesi, S. Pinna and I. Turnu
Dipartimento di Ingegneria Elettrica ed Elettronica
Università di Cagliari - piazza d'armi, 09123 Cagliari, Italy

received 5 June 2006; accepted in final form 19 October 2006
published online 16 November 2006

PACS. 89.75.-k – Complex systems.
PACS. 89.75.Da – Systems obeying scaling laws.
PACS. 61.43.Hv – Fractals; macroscopic aggregates (including diffusion-limited aggregates).

Abstract. – A large number of real networks are characterized by two fundamental properties: they are small world and scale-free. A recent paper demonstrated that the structure of many complex networks is also self-similar under a length-scale transformation, and calculated their fractal dimension using the "box counting" method. We studied nine large object-oriented software systems, finding that the graphs associated to these networks are self-similar. We also studied the time evolution of the fractal dimension during system growth, finding a significant correlation between the fractal dimension and object-oriented complexity metrics known to be correlated with software fault-proneness. Thus, in software systems the fractal dimension could be considered as a measure of internal complexity, and consequently of the system quality.

Introduction. – Small-world and scale-free properties have been found in a large number of real networks [1]. A recent study [2] has found that the structure of these networks is often also self-similar, and it is possible to calculate their fractal dimension using the box-counting method. This method consists in covering the entire network with the minimum number of boxes N_B of linear size ℓ_B. If the number of boxes scales with the linear size ℓ_B following a power law (see eq. (1)), then d_B is the fractal dimension, or box dimension, of the graph [3]:

$$N_B\left(\ell_B\right) \sim \ell_B^{-d_B}. \tag{1}$$

Song *et al.* applied this procedure to several different real networks: a part of WWW, a social network, the biological networks of protein-protein interaction found in *E. Coli* and *H. Sapiens*, etc., finding consistent self-similar behavior. On the other hand, they analyzed also the Barabasi-Albert growth model of complex networks [4] —which introduces the concept of preferential attachment— finding that graphs generated in this way, as well as the Internet, are not self-similar(1). In our research, we studied whether networks associated to software systems have a fractal dimension associated, and in this case what is its value. In particular, we considered object-oriented (OO) software systems, OO programming being the most widely used software technology.

Object-oriented systems. The basic building block of OO programming is the *class*, composed of a data structure and of procedures able to access and process these data. The data structure is made up of fields (instance or class variables) that represent the state of an object.

(1)See Supplementary Information of paper [2].

A class has also a behavior expressed in terms of methods that represent the procedures able to access and process the data structure. Classes may be defined at various levels of complexity, and are related across different kinds of binary relationships, such as inheritance, composition and dependence. A class may inherit from another class both state and behavior through an inheritance relation; the former is called subclass, the latter superclass. Declaring a field of a given class within another class leads to a composition relationship between the classes. A possible definition of dependence of class A on class B (among others found in the literature that are in practice equivalent) is that A depends on B if A accesses a field or calls a method defined in B. All these relationships are directed, in the sense that they involve a class that supplies some service (server class), and a class that uses this service (client class). When developing an OO software system, software engineers write classes and link them with other classes of the system. The interactions between all the classes yield the program functionality.

Analyzing the source code of an OO system, it is possible to build its *class graph* —a graph whose nodes are the classes, and the graph edges represent directed relationships between classes. In this graph, the in-degree of a class is the number of edges directed toward the class, and is related to the usage level of this class in the system. On the other hand, the out-degree of a class is the number of edges leaving the class, and represents the level of usage the class makes of other classes in the system. It has already been shown that software networks are small-world and scale-free [5–7]. The in-degree distributions are power laws with exponent $\gamma \approx 2.5$. The out-degree distributions are more controversial. Some authors found power law behavior [5–7], whereas others found a lognormal behavior [8].

Software networks have not yet been studied in order to find fractal properties in their topology. This perspective could be very interesting for software engineering, because one of the most desirable properties of OO architecture is *modularity*, and Song *et al.* in a recent paper [9] showed that fractal networks are characterized by a modular hierarchical structure. This implies that a well-engineered OO system, being modular, should show self-similarity.

In this paper we investigate on the fractal properties of OO software systems, and demonstrate that a fractal dimension exists also for this kind of networks, as expected due to the above considerations. We also investigate on the relationship between the fractal dimension computed for various OO systems, and standard metrics related with software quality.

The Chidamber and Kemerer Suite of object-oriented quality metrics. The definition and the validation of a suite of quality metrics for OO software systems plays an important role in the software engineering research field. The most known suite of OO metrics is that of Chidamber and Kemerer (CK). The CK suite is composed of six metrics: Weighted Methods per Class (WMC), Coupling Between Object Classes (CBO), Depth of Inheritance Tree (DIT), Number of Children (NOC), Response for a Class (RFC), Lack of Cohesion of Methods (LCOM) [10].

Some CK metrics measure internal properties of classes, for example the number of methods (WMC) and their cohesion. Cohesive methods share at least one of the class fields with another method. The lack of cohesion metric (LCOM) is the difference between the number of non-cohesive methods pairs and the number of cohesive pairs.

CBO is a count of the number of other classes which a given class is coupled to, hence denotes the class dependency on other classes in the system. CBO metric for a given class is strictly related to the out-degree of the class node in the class graph, where links represent dependences between classes. RFC for a given class A is computed as the sum of the number of methods defined in A and of the cardinality of the set of methods called by them and belonging to external classes. The number of elements of this set is equal to the weighted number of out-links of class-A node, if the link between class A and another class B is weighted with the number of different methods of B called by A (weighted dependence).

NOC and DIT measure properties of the inheritance tree. The NOC metric of a class is the number of directed subclasses of the class. The DIT metric of a class is the number of nodes (classes) to be crossed to reach the root of the inheritance tree.

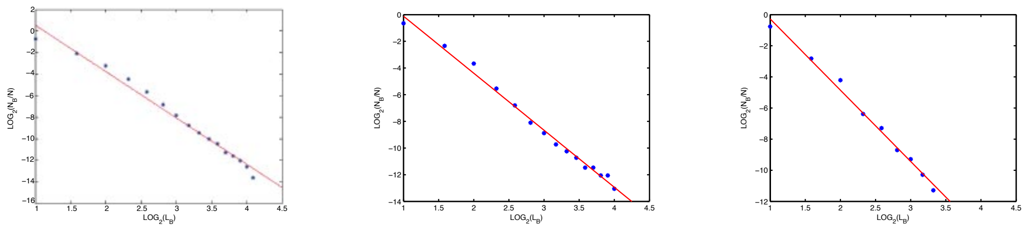
CK metrics have been widely validated in the literature, [11–14]. Many researchers found a correlation between some CK metrics and the fault-proneness of classes. Others found that some CK metrics are correlated with the effort needed for maintaining the classes. Among CK metrics, CBO is certainly the most validated, and almost always found highly correlated with system fault-proneness. High coupling negatively affects the system quality because if a bug is injected into a class this may potentially infect all the classes that depend on it. So the class coupling must be kept as low as possible. Other CK metrics which were often found to be correlated with system faults are WMC and RFC.

For each software system studied, we calculate both the fractal dimension and CK metrics. We repeat these measures for different versions of the system and observe similar patterns between the evolution of the fractal dimension and the evolution of the system quality, expressed in terms of some CK metrics.

Fractal dimension of software systems. – We studied the class graphs of the following nine OO software systems, whose source code is freely available on the Internet: JDK, Eclipse, Eclipse Birt, Netbeans, JBoss, Visual Works, SmallTalk/X, Squeak and GlassFish([2]). Using the same procedure followed by Song *et al.* [2], we calculated the fractal dimension d_B for each of the considered graphs. All these graphs revealed a self-similar structure. Note that the box-counting algorithm defined in [2] works only for non-directed graphs, because the distance between two nodes taken as the shortest path between the nodes would not satisfy the expected commutative property in the case of directed graphs. For this reason, we did not consider link orientation in the computation of the fractal dimension of the graphs. The coverage algorithm used to tile the entire network with boxes of size ℓ_B was devised after many tests with different possible algorithms. While a proof of its optimality is out of discussion, being the optimization problem in the NP-complete complexity class (see footnote ([1])), it yielded the best tiling we were able to find, and on the networks studied in [2] it yielded the same results reported there. It is also computationally quite efficient, because tiling a network with 10000 nodes and 140 000 links, varying ℓ_B from 2 to 16 with steps of one, takes less than $30''$ on a 3 GHz PC. It may be described as follows. For $\ell_B = 2$: 1) a node is selected randomly among those with smallest degree and is used as a seed; 2) the nodes having a distance less than ℓ_B from the seed and from each other are assigned to a box, and the nodes belonging to the box are burned; 3) steps 1) and 2) are repeated considering only non-burned nodes, until all nodes are assigned to their respective boxes. In step 1) we choose the nodes with smallest degree because we found that, starting with the nodes with highest degree (the *hubs* of the network), we tend to find in the beginning some boxes with very many nodes, but then we leave out many other poorly connected nodes. In the end, the value of N_B tends to be higher when following the latter procedure. The proposed algorithm was also tested to work better than simple random choice of the seed node of step 1).

For $\ell_B > 2$, the tiling starts from the previous tiling made for $\ell_B - 1$ and proceeds in the following way: 1) a box is selected randomly among those with smallest cardinality; 2) all boxes adjacent to the selected one are probed, trying to merge them with it; if the merging yields a new box whose nodes have a distance less than ℓ_B from each other, it is retained and the two merged boxes are burned; 3) steps 1) and 2) are repeated considering only non-burned

([2])The Web sites of these projects are, in the same order of citation: `java.sun.com`, `www.eclipse.org`, `www.eclipse.org/birt`, `www.netbeans.org`, `www.jboss.org`, `smalltalk.cincom.com`, `www.exept.de`, `www.squeak.org`, `glassfish.dev.java.net`.



(a) *Eclipse 2.1.3* (b) *JDK1.5.0* (c) *VisualWorks 7.3*

Fig. 1 – (a)-(c) Box-counting analysis of software networks. The log-log plot of N_B *vs.* ℓ_B reveals a self-similar structure for (a) Eclipse, (b) JDK, (c) VisualWorks. Similar plots are observed also for the remaining systems.

boxes of the previous tiling, until all nodes are assigned to their respective new boxes. Starting from the previous tiling is by far more time-efficient, and on average yields a lower value of N_B, than starting over again for each new value of ℓ_B.

The number of boxes N_B needed to tile the entire network is computed for different values of ℓ_B. If eq. (1) holds, then the exponent of the power law d_B is the fractal dimension.

Figure 1 shows the log-log plots of N_B *vs.* ℓ_B for three of the studied systems. Similar plots are observed also for the other systems indicating that all the systems we analyzed fairly fit a straight line, and hence there is a power law dependence between N_B and ℓ_B. The values of d_B range from 3.7 to 5.1, and are similar to values reported in [2] for the graphs studied there.

For each system, we repeated 50 times the calculation of the fractal dimension d_B, because our box-counting method is not deterministic, and its results depend on the starting point and the random number generator seed. In table I we report mean and standard deviation of the self-similar exponent d_B for all the analyzed systems.

As regards the statistical nature of these computations, we performed a set of Kolmogorov-Smirnov tests [15] to find the probability distributions that better fit our data. The test results (with a confidence of 99%) showed that the values found of *fractal dimension* are distributed according to a normal distribution. The low values of the standard deviation denote that the box-counting algorithm we used is quite robust.

Evolution of the fractal dimension. – Software systems evolve in time, as new features are added, and bugs are fixed. The corresponding graphs also evolve, both in terms of number of nodes (classes) and links (relationships between classes). For instance, JDK evolved from about 600 nodes up to 8000. The evolution of some software systems may be easily studied because programmers use version control systems for the source code, as for instance CVS [16] or Subversion [17]. In our study, we were able to analyze the evolution of five systems, whose

Table I – *Average results, over* 50 *runs, representing the fractal dimension of the networks studied. Standard deviations are reported in parenthesis.*

Networks	d_B	Networks	d_B
JDK1.5.0	4.24 (0.015)	Jboss 20060113	3.78 (0.005)
Eclipse 2.1.3	4.31 (0.012)	Visual Works 7.3	4.54 (0.021)
Netbeans 5.0	4.43 (0.019)	SmallTalk/X 5.25	4.07 (0.026)
Eclipse Birt 20060213	3.71 (0.021)	Squeak 3.5	5.14 (0.027)
GlassFish 20060213	3.79 (0.007)		

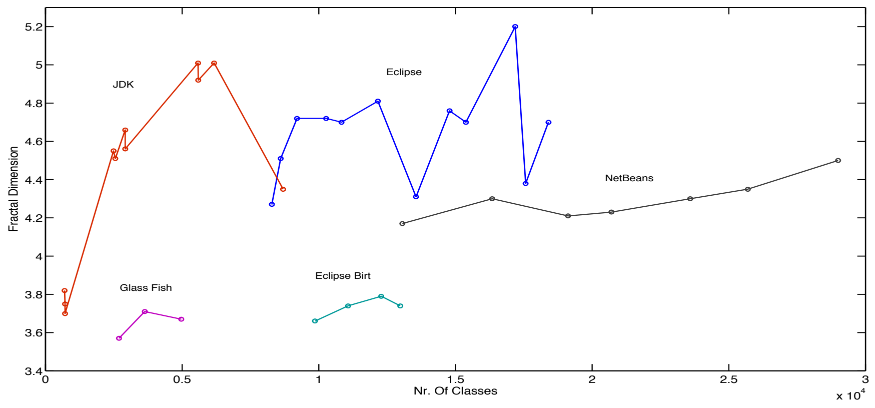


Fig. 2 – Fractal dimension for different versions of the analyzed systems, as a function of the number of classes of each version.

repositories are freely available on the Web: Eclipse (12 versions), JDK (11 versions), Netbeans (7 versions), GlassFish (4 versions), Eclipse Birt (3 versions).

Figure 2 shows how the fractal dimension of these systems evolved as a function of the number of classes of each version of the system, and hence approximately of time.

We see that there are different change patterns of d_B. Netbeans presents a slow, steady increase of d_B, but for an initial version. Eclipse and JDK trends are much more irregular. JDK presents big jumps, corresponding to versions with major upgrades and major increases in the number of classes. These jumps are basically upward, but for the last one. We will discuss this behavior in more detail in the next section. Eclipse shows the most irregular behavior, but overall its upward trend is quite slight. GlassFish and Eclipse Birt have a fractal dimension that seems to slowly increase with the number of classes, but we could analyze only a very few versions. The general trend, however, is of an overall increase of the fractal dimension with the size of the system.

As regards other topological properties of the evolving software systems, we analyze also the nodes in-degree distributions, which patently follow a power law. We evaluated the exponent γ of these distributions using the Hill estimator [18]. The values of γ tend to stay quite constant during system evolution. For instance, all Eclipse versions have γ that ranges from 2.4 to 2.6, while Netbeans versions have γ in the range 2.1–2.3. For this reason, also considering that the γ exponent cannot be calculated with high precision, we did not further analyze its change patterns through the different versions of software systems.

Correlation between software metrics and fractal dimension. – We analyzed together the evolution of the fractal dimension and the evolution of CK metrics for the software systems with a significant number of versions, namely JDK, Eclipse and Netbeans. For each of these systems we analyzed also the correlations among the values of CK metrics and the fractal dimension. Since CK metrics refer to single classes, while the fractal dimension is a global measure of the system, we computed the mean value of each metric for each version of the analyzed system. CK metrics distributions for a software system are usually lognormal or even power law. We focused our attention to CBO and RFC metrics, which are the most correlated with quality and complexity of a software system, and found that these metrics

Table II – *Correlation coefficients between the CK metrics and the fractal dimension for all the considered versions.*

Network	CBO-d_B	σ_{CBO}-d_B	RFC-d_B	σ_{RFC}-d_B	DIT-d_B	NOC-d_B	LCOM-d_B	WMC-d_B
JDK	0.95	0.92	0.92	0.83	−0.51	−0.50	0.19	0.68
Eclipse	0.93	0.92	0.94	0.93	0.21	0.01	0.33	0.38
Netbeans	0.75	0.82	0.92	0.44	−0.76	−0.78	−0.66	−0.05

follow a lognormal distribution. We estimated the parameters of these distributions for each version of the analyzed system. In particular, the standard deviation of the lognormal is an index of the "fatness" of the tail, and thus of the number of classes with high values of the metrics. Thus, for CBO and RFC metrics we calculated the correlation of the fractal dimension with both their means and standard deviations, σ_{CBO} and σ_{RFC}. The correlation coefficients between CK metrics and fractal dimension are shown in table II.

We systematically found high correlations of both mean and standard deviation of CBO and RFC metrics with the fractal dimension. The other CK metrics do not show correlation with d_B, but this is not unexpected, because WMC and LCOM measure internal properties of the classes, whereas the fractal dimension takes into account the relationships among them; moreover, NOC and DIT measure properties of the inheritance graph, while the fractal dimension is calculated on a graph whose links mostly represent dependence relationships.

The strong correlation between the fractal dimension d_B and the mean value of CBO is not unexpected. At a structural level, an increase of CBO yields a reduction of the number of boxes N_B of a given dimension ℓ_B because classes are more coupled, and it is more likely to find boxes with a high number of nodes. Finally, if eq. (1) holds, a reduction of N_B leads to an increase of d_B. Note that the mean value of CBO is equal to the average number of out-links per node, which in a graph is equal to the average number of in-links. Thus, the mean value of CBO equals one half the average number of edges per node in the undirected class graph used to calculate the fractal dimension (not considering edges due to inheritance, that account for less than 1% of the total). Similar considerations apply to the weighted directed graphs used to compute RFC. So, despite the fact that in-links and out-links of a graph are very different from a software engineering point of view, their averages are closely linked to those of the undirected graph used to compute d_B. As regards standard deviations, we observe that the fractal dimension is strongly correlated also with σ_{CBO} and, to a lesser extent, with σ_{RFC}. This empirical finding supports the hypothesis that d_B is correlated with system complexity, because a system with fatter tails in the distributions of CBO and RFC has more classes with very high values of these metrics, and can thus be considered more complex and error-prone.

Table III reports in detail the metrics computed for each version of JDK. We report the number of classes, the mean and standard deviations of CBO and RFC, and the fractal dimensions. As you can see, the major releases were versions 1.2.1, 1.4.0 and 1.5.0, characterized by a major increase in the number of classes of the system. All these versions present a significant

Table III – *Correspondence between the CK metrics and the fractal dimension.*

JDK	Classes	CBO (σ_{CBO})	RFC (σ_{RFC})	d_B	JDK	Classes	CBO (σ_{CBO})	RFC (σ_{RFC})	d_B
1.1.6	699	2.26 (28.1)	12.80 (1123)	3.8	1.3.1	2918	4.21 (46.7)	17.28 (1265)	4.6
1.1.7	717	2.25 (28.6)	12.65 (1119)	3.7	1.4.0	5582	4.47 (56.7)	17.51 (1241)	5.0
1.1.8	717	2.26 (29.2)	12.70 (1126)	3.7	1.4.1	5590	4.49 (58.6)	17.57 (1285)	4.9
1.2.1	2488	4.11 (43.6)	17.44 (1271)	4.5	1.4.2	6170	4.70 (62.3)	17.86 (1340)	5.0
1.2.2	2558	4.19 (45.7)	17.52 (1294)	4.5	1.5.0	8696	4.25 (60.6)	17.16 (1121)	4.2
1.3.0	2917	4.20 (46.5)	17.25 (1262)	4.7					

increase in the values of the considered metrics, but the last one (1.5.0). Note that release JDK 1.5.0 is considered a very important new release, not only because the number of system features (and hence the number of classes) is substantially increased, but also because the system underwent a major *refactoring*, a software development activity aimed to lower duplications, to get a better structure of the system, and in the end to reduce its "disorder" [19]. This is reflected in the reduction of CBO and RFC values, of their standard deviations, and also in a reduction of the fractal dimension. So, we can infer that an increase of d_B is associated with an increase of the complexity of the system, that is consequently more difficult to maintain. The other systems analyzed, Eclipse and Netbeans, confirm these results showing similar correlations between fractal dimension and CBO and RFC metrics. The evolution of the fractal dimension of Eclipse shows the most peculiar behavior, alternating strong increases and decreases of its value to intervals where it is practically constant, despite the overall tenfold increase in the number of classes (see fig. 2). Note that Eclipse project is known for the strong, ongoing refactoring that characterizes its development. If the assumption that the fractal dimension is associated with the complexity of the system holds, we may deduce that this constant refactoring could be the reason behind the fact that Eclipse's fractal dimension does not substantially increase, despite the strong increase of its number of classes along the twelve observed versions.

* * *

This work was supported by MAPS (Agile Methodologies for Software Production) research project, of FIRB research fund of MIUR; contract number: RBNE01JRK8.

REFERENCES

[1] Albert R. and Barabasi A.-L., *Rev. Mod. Phys.*, **74** (2002) 47.
[2] Song C., Havlin S. and Makse H. A., *Nature*, **433** (2005) 392395.
[3] Feder J., *Fractals* (Plenum Press, New York) 1988.
[4] Barabasi A.-L. and Albert R., *Science*, **286** (1999) 509.
[5] Myers C. R., *Phys. Rev. E*, **68** (2003) 046116.
[6] Valverde S., Ferrer-Cancho R. and Solé R. V., *Europhys. Lett.*, **60** (2002) 512.
[7] Valverde S. and Solé R. V., *Hierarchical Small Worlds in Software Architecture* (Santa Fe Inst. Working Paper, SFI/03-07-044) 2003.
[8] Concas G., Marchesi M., Pinna S. and Serra N., *Power-laws in a Large Object-Oriented Software System*, submitted to *IEEE Trans. Software Eng.* (2006).
[9] Song C., Havlin S. and Makse H. A., *Nat. Phys.*, **433** (2006) 275281.
[10] Chidamber S. R. and Kemerer C. F., *IEEE Trans. Software Eng.*, **20** (1994) 476.
[11] Li W. and Henry S., *J. Syst. Software*, **23** (1993) 111.
[12] Basili V., Briand L. and Melo W., *IEEE Trans. Software Eng.*, **22** (1996) 751.
[13] Subramanyam R. and Krishnan M. S., *IEEE Trans. Software Eng.*, **29** (2003) 297.
[14] Gyimo T., Ferenc R. and Siket I., *IEEE Trans. Software Eng.*, **31** (2005) 897.
[15] David Sheskin, *The Handbook of Parametric and Nonparametric Statistical Procedures* (CRC Press) 2003.
[16] `http://www.nongnu.org/cvs/`.
[17] `http://subversion.tigris.org/`.
[18] Newman M., *Contemp. Phys.*, **46** (2005) 323.
[19] Fowler M., Beck K., Brant J., Opdyke W. and Roberts D., *Refactoring: Improving the Design of Existing Code* (Addison-Wesley, Reading, Mass.) 1999.