Trends in Object-Oriented Software Evolution: Investigating Network Properties

Alexander Chatzigeorgiou and George Melas
Department of Applied Informatics
University of Macedonia
Thessaloniki, Greece
achat@uom.gr, melas@uom.gr

***Abstract*—The rise of social networks and the accompanying interest to study their evolution has stimulated a number of research efforts to analyze their growth patterns by means of network analysis. The inherent graph-like structure of object-oriented systems calls for the application of the corresponding methods and tools to analyze software evolution. In this paper we investigate network properties of two open-source systems and observe interesting phenomena regarding their growth. Relating the observed evolutionary trends to principles and laws of software design enables a high-level assessment of tendencies in the underlying design quality.**

***Keywords*-software evolution; network analysis; object-oriented design**

I. INTRODUCTION

Following the unprecedented rise of social networks in an extremely short period of time [1], there has been considerable interest in studying the graph structures arising in these settings [7]. Software systems and especially the structure of object-oriented ones have also been the focus of research efforts, seeking to identify properties such as scale-freeness and small-world phenomena, based on individual snapshots of the corresponding networks [3], [9]. However, given that most software systems evolve over a number of versions, interesting phenomena arise and are worth of investigating by observing how fundamental network properties vary with time.

The architecture of object-oriented systems can be effectively captured by directed graphs, where classes are represented as nodes, while edges correspond to the selected type of relationship to be modeled. Studying the way in which graphs change when adaptive or corrective maintenance is performed, can reveal interesting evolutionary trends and quantitative results.

While in social network analysis this information might be useful to understand the nature of the resulting groups of people (e.g. to study collaborating software developers), in the field of software evolution analysis such data might reveal the presence of design problems, the application of refactorings in the software history or the formulation of "evolution-oriented" design rules and patterns. The ultimate goal is to interpret macroscopic phenomena related to software development (such as software ageing and social influence among developers) by associating them to the examined properties at the microscopic node level.

Graph Representation. Object-oriented systems can be naturally represented as graphs, where nodes correspond to classes and edges to relations between classes [3]. In the context of this study, class friendship relationships are modeled according to the Law of Demeter [8]. In other words, an edge between two classes implies the presence of any of the allowed alternatives for the invocation of methods in "friendly" classes, i.e. references to the target class which are either attributes, local variables to which instances of the target class are assigned, method parameters and return types of the target class type.

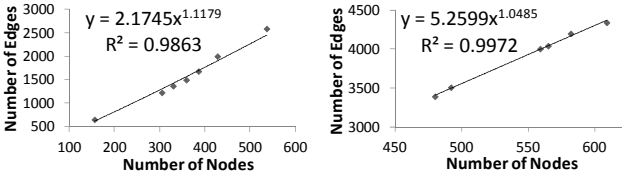
DataSets. In order to study the evolution of network properties, several versions of two open source projects (written in Java) have been selected, namely Weka and JFreeChart. Weka is a workbench containing a collection of machine learning algorithms for data mining tasks, for which development started in 1997. JFreeChart is a chart library which has been constantly evolving since 2000. 7 versions of Weka and 6 versions of JFreeChart have been analyzed.

II. SYSTEM DENSIFICATION

For a broad range of networks, such as citation, router and author-paper affiliation graphs it has been shown that they follow a kind of densification law [6]. This means that graphs are becoming denser during their evolution in the sense that the number of edges increases at a higher rate than the number of nodes. Formally stated, if $E(t)$ and $N(t)$ represent the number of edges and nodes at a certain time point t, graphs evolve according to the relation $E(t) \propto N(t)^{a}$, where the densification coefficient α ranges between 1 and 2.

In the context of object-oriented systems where edges represent "friend" relationships among classes, a dense graph implies high coupling. Figure 1 shows plots of the number of edges $E(t)$ versus the number of nodes $N(t)$, for the examined systems and all software versions. The exponent of the corresponding densification law is also shown. As it can be observed, in both cases α is slightly higher than 1 indicating a small deviation from a linear growth of the systems.

An interpretation of this observation is that as the examined systems evolve, the exploitation of existing methods (to access their functionality) and the addition of new methods is a more often maintenance activity than the addition of new classes. Assuming that each new software version is equipped with additional functionality, it can be claimed that a densification coefficient that is significantly higher than one, implies non-conformance to the Open-Closed Principle according to which the introduction of new functionality should be implemented by adding new modules rather than modifying existing ones.



(a) Weka ($\alpha = 1.11$) (b) JFreeChart ($\alpha = 1.05$)

Figure 1. Number of edges versus number of nodes.

III. COMMUNITY GUIDED ATTACHMENT

Should we expect that classes in an object-oriented system macroscopically behave like people in a social network - in the sense that classes tend to interact mostly with classes which belong to a similar community? The corresponding notion in social networks is termed *homophily* [4], and states that we tend to be similar to our friends. The mechanisms that lead to similar characteristics among friends have been extensively studied and are mainly *selection* (the tendency of people to form connections with others who share similar characteristics) and *social influence* (which refers to the way people modify their behavior to bring their activities into alignment with the activities of their associates). Under this perspective, it is natural that people that share foci around the same club form ties more easily than people of different clubs.

Obviously classes do not select their own collaborators and do not modify their behavior for social reasons. But the designers of classes make them interact with other classes to access functionality or data and it would be reasonable to assume that classes exchange messages with classes that belong to similar conceptual groups. For example, classes in the business logic part of a software system tend to interact with other classes of the same component in order to carry out their computations. This is mostly evident from the fact that such classes are placed in distinct "communities" which in the case of software organization are packages or name spaces. Regarding the evolution of software, pairs of classes belonging to the same community (package) would be expected to form relations (in most cases associations) more frequently, than classes belonging to different communities or classes that share membership in a larger community (e.g. a package at a higher level).

Formally, the structure of packages-within-packages of a software system can be represented as a tree. Classes are the leaves of the tree while internal nodes represent packages. The parent of each node indicates the package in which the corresponding element is nested. The tree in Figure 2 represents the structure of packages-within-packages for part of JFreeChart software. `org.jfree` is the root package housing among others the `chart`, `gui` and `data` packages.

A Community Guided Attachment model of growth [6] is based on the intuitive assumption that cross-package links should be harder to form than intra-package links between classes. Moreover, it would be reasonable to believe that the difficulty in forming relations between classes should become higher as the distance between packages increases.

The most appropriate way to define the distance between two classes c_1 and c_2 is to employ the standard tree distance

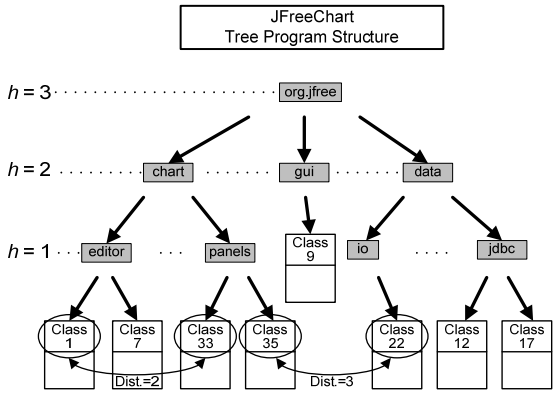
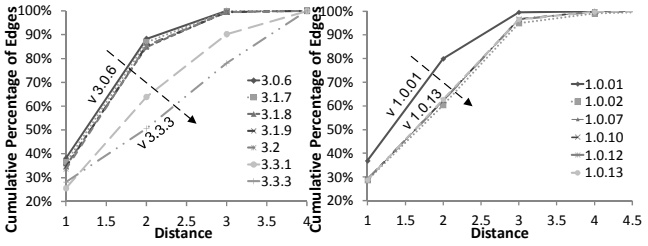


Figure 2. Tree example representing packages-within-packages structure.

$h(c_1, c_2)$. According to this measure, the distance between two leaf nodes is the height of their least common ancestor. (The height of a node is the length of the longest path to a leaf from that node). The height of internal nodes/packages is shown in the left hand side of Figure 2).

Community Guided Attachment would manifest itself as a large percentage of links among the classes belonging to the same package. The number of relations between classes of different packages should become less as the distance of the corresponding packages increases. To investigate whether community guided attachment applies for the systems under study we show in Figure 3 the cumulative plot of the percentage of associations between classes at several distances, for all versions of Weka and JFreeChart.



(a) Weka (b) JFreeChart

Figure 3. Distribution of class distance

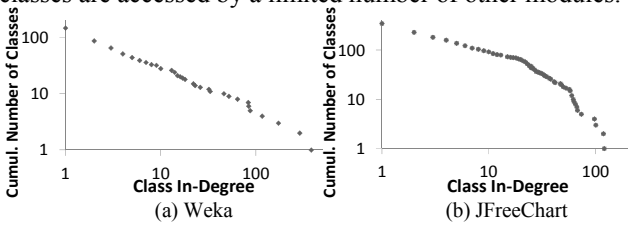
For example, in the last examined version of Weka (rightmost curve), only around 30% of the links connect classes which are at distance 1, i.e. belong to the same package. As a result, the largest portion of associations are links that connect classes residing in different packages. It is interesting to note that a very large number of associations are formed between classes that have a distance of 2, 3 or 4, indicating that community guided attachment does not hold for this software system. Regarding the tendency over the number of software versions (see arrows on the diagrams) it is evident that with the passage of time more and more links are formed between classes which are further apart. In other words, even if it exists, the presence of the community guided attachment model gets weaker over time. By examining in source code the actual associations that have been created during each transition to a new version, cross-community links are formed mainly because new classes are placed in newly created packages.

According to Leskovec et al. [6], the existence of a community guided attachment model leads to networks that are becoming denser over time. The fact that in the examined systems, the community guided attachment model is hardly present can explain the relatively low densification coefficients derived in the analysis of Section II.

IV. PREFERENTIAL ATTACHMENT

One of the most extensively studied issues of systems that can be represented as networks is related to the mechanisms that lead to scale-free properties. Scale free phenomena show up statistically in the form of a power law and are quite frequent in technological, social and biological networks. Many researchers attribute the generation of scale-free networks to the existence of a preferential attachment model [2] which postulates that when a network evolves by having new nodes join the existing structure, the number of new links attracted by each destination node is proportional to the destination's degree. This mechanism, also known as "rich-get-richer" rule, for an object-oriented system implies that God classes having a large in-degree, are more probable to be the destination nodes of new links when classes are added to the system.

Since the network of "friendly" classes according to the Law of Demeter constitutes a directed graph, we can plot the in-degree distribution of the classes. A high-in degree indicates a heavily used class with which other classes communicate, either to access its functionality or its state. As it is reasonable to expect, Figure 4 (log-log plot) shows that in-degree exhibits a heavy-tailed distribution, where the largest percentage of classes have a low in-degree (between 1 and 10) and very few classes have in-degree that exceeds 100. This power-law distribution implies that few classes provide services to a large number of clients, while most classes are accessed by a limited number of other modules.

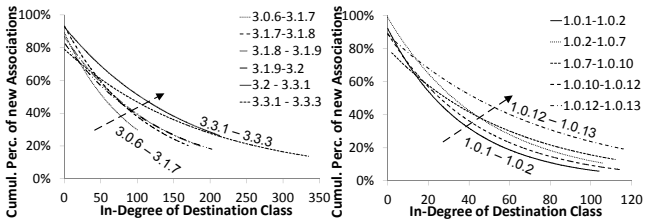


(a) Weka (b) JFreeChart

Figure 4. In-Degree distribution

Several models have been proposed in the literature to investigate the primary causes that lead to power-law phenomena. As already mentioned, among the proposed models preferential attachment is probably the most widely accepted since it is reasonable to assume that already important nodes in terms of the number of connections to them or "hubs", act as attractors for new members that join an existing network.

To investigate the presence of preferential attachment and more importantly its evolution as software matures, the edge attachment by degree should be studied. To this end, we calculate the cumulative percentage of new associations formed in each new version of the examined software versus the degree of the destination node. The results are depicted graphically in Figure 5.



(a) Weka (b) JFreeChart

Figure 5. Evolution of new associations per destination degree.

As it can be observed, for the first version of Weka, around 50% of the new associations added in that version are attached to nodes having an in-degree 50 or higher, while for JFreeChart around 50% of the new associations are reaching nodes with an in-degree 20 or higher. This certainly implies the presence of a scale-free phenomenon that manifests itself as a long tail in the cumulative frequency plot and would be clearer in a log-log scale where the distribution would appear as a line of slope $-k$, where k is the scaling index [2].

However, the most striking observation in these plots is that preferential attachment becomes more intense as software evolves. With the passage of versions a larger percentage of new classes are linked to higher in-degree classes. For example, on the transition to the last examined version of Weka, more than 50% of the new links are attached to classes having a degree higher than 100, while a remarkable 25% are attached to classes having in-degree higher than 250. A similar trend is observed for JFreeChart.

Such observations, if validated by studies on other systems, should be alarming for the maintainability of the corresponding projects. A gradually increasing compliance to the rich-get-richer rule implies that God classes are becoming even more worrisome, incorporating an increasing number of either data members or methods. With the exception of utility classes providing general services to many clients, the gradual growth of already large classes with the simultaneous dependence of numerous clients on them should be an indicator to the development team that they warrant much more maintenance effort and attention.

V. SMALL WORLD PHENOMENA

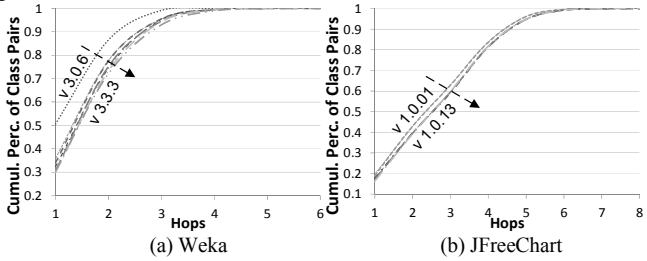
A network (usually a social one) is said to exhibit the *small-world phenomenon* if any two nodes in the network have a high probability of being associated through a short path of intermediate nodes [5]. This is popularly known as *six degrees of separation* according to Milgram's [10] experiment which states that any two people on this planet can be connected via an average number of six steps (intermediate acquaintances). Research findings frequently suggest that the phenomenon is pervasive in technological (Web pages, routers) and social networks (Facebook, Flickr, LinkedIn). Another famous network exhibiting small world phenomena is the co-authorship graph where edges connect mathematicians who have jointly authored a paper. A mathematician's *Erdös number* is the distance from him to the legendary mathematician Paul Erdös [4]. Suprisingly, most mathematicians have Erdös numbers of at most 4 or 5.

The existence of the small-world phenomenon is usually associated with the small diameters that the corresponding

networks have. According to the model proposed by Watts and Strogatz [11] this property stems from *homophily* (the tendency of nodes to connect to other nodes that are similar) and the presence of *weak ties* (edges that link together distant nodes of a graph) [4].

In the object-oriented world one can easily observe that both aforementioned network properties are present: Classes tend to form associations with other classes that are similar in terms of their functionality and conceptual meaning (homophily) and thus edges link classes that are in the same "neighborhood" of the system (e.g. package). At the same time, as it has been demonstrated by the non-conformance to the community guided attachment model, links are also formed between classes that are distant in terms of the system structure (*weak ties*). If these assumptions are correct, graphs formed by "friendly" classes are expected to have low average diameters and exhibit the small world phenomenon.

As in the previous investigations, our goal is to examine the tendency (if any) of the corresponding phenomenon as software systems evolve over time. To this end, Figure 6 shows hop plots over time, illustrating the cumulative percentage of class pairs which are *h* hops apart (for all examined versions). The fact that 100% of class pairs are at most 4 and 7 hops apart (for Weka and JFreeChart respectively) indicates the presence of the small world phenomenon in the examined software systems.



(a) Weka (b) JFreeChart

Figure 6. Cumulative Percentage of class pairs which are *h* hops apart

The trend captured in the diagrams of Figure 6 (indicated by the arrow pointing from the oldest to the newest examined version) points to an interesting observation: The small world phenomenon tends to become less intense over time, since for each new version, a smaller percentage of class pairs are *h* hops apart, for any given *h* and the furthest distance in term of hops becomes larger.

According to Leskovec et al. [6], networks exhibit shrinking diameters over time (which in turn lead to small world phenomena) when they follow a "forest fire" growth model. The "forest fire" model imitates a process where a new node arrives at a network, chooses a target node to which it is linked and then forms links to the target node's friends, while this process is assumed to continue recursively (like a fire spreading from one tree to its neighboring trees).

However, this is not how object-oriented systems evolve: Adding a new class to a system (and associating it to one or more target classes), does not mean that the newly introduced class will later on form associations to the "friends" of the target classes. On the contrary, since the target classes provide services (which most usually employ their surrounding friend classes) there is no need for the new class to access the targets' friend classes. From a design perspective this would lead to needless increase of coupling among classes affecting negatively system maintainability.

As a result, the forest-fire model does not make sense for the evolution of object-oriented systems. The absence of a forest-fire model does not force shrinking diameters for the class graphs and this fact provides a justification on why small world phenomena tend to weaken over time. Indeed, the diameter of the graph corresponding to Weka increased from 4 to 6 between the examined versions, while the diameter for JFreeChart increased from 7 to 8.

VI. Conclusions and Future Work

The analysis of software as it evolves to accommodate new requirements in order to reveal the underlying trends in its growth can be a challenging task, especially for large systems with hundreds of modules and thousands of relations among them. Network analysis techniques that have been employed to study social systems can provide valuable insight into evolution phenomena which are related to the quality properties of the corresponding software systems.

A major difference between software systems and social networks is that we can intentionally modify their structure as part of the adaptive, corrective or preventive maintenance. A line of future research would be to investigate the impact of design improving activities, such as refactorings, on the evolutionary trends that have been discussed in this paper. Furthermore, other important properties at the node and network level such as centrality, bridging and clustering, might provide interesting views into the growth of software.

References

[1] M. Anderson, "The Data: Six Billion Friends," IEEE Spectrum, vol. 48, June 2011, pp. 80.

[2] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," Science, vol. 286, October 1999, pp. 509-512.

[3] A. Chatzigeorgiou, N. Tsantalis, and G. Stephanides, "Application of Graph Theory to OO Software Engineering," Proc. IEEE Work. Interdisciplinary Software Engineering Research (WISER 06), May 2006, pp. 29-35.

[4] D. Easley and J. Kleinberg, Networks, Crowds, and Markets: Reasoning about a Highly Connected World, Cambridge University Press, 2010.

[5] J. Kleinberg, "The Small-World Phenomenon: An Algorithmic Perspective," Proc. ACM Symposium on Theory of Computing (STOC 00), May 2000, pp. 163–170.

[6] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations," Proc. ACM Int. Conf. Knowledge Discovery in Data Mining (KDD 05), August 2005, pp. 177-187.

[7] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic Evolution of Social Networks," Proc. ACM Int. Conf. Knowledge Discovery in Data Mining (KDD 08), August 2008, pp. 462-470.

[8] K. Lieberherr and I. Holland, "Assuring good style for object-oriented programs," IEEE Software, vol. 6, September 1989, pp.38–48.

[9] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," ACM Transactions on Software Engineering and Methodology, vol. 18, September 2008, pp. 1-26.

[10] S. Milgram, "The small world problem," Psychology Today, vol. 1, May 1967, pp. 60-67.

[11] D. J. Watts and S. H. Strogatz, "Collective dynamics of `small-world' networks," Nature, vol. 393, June 1998, pp. 440-442.