Forecasting Java Software Evolution Trends Employing Network Models

Theodore Chaikalis, *Member, IEEE* and Alexander Chatzigeorgiou, *Member, IEEE*

Abstract—The evolution of networks representing systems in various domains, including social networks, has been extensively studied enabling the development of growth models which govern their behavior over time. The architecture of software systems can also be naturally represented in the form of networks, whose properties change as software evolves. In this paper we attempt to model several aspects of graphs representing object-oriented software systems as they evolve over a number of versions. The goal is to develop a prediction model by considering global phenomena such as preferential attachment, past evolutionary trends such as the tendency of classes to create fewer relations as they age, as well as domain knowledge in terms of principles that have to be followed in object-oriented design. The derived models can provide insight into the future trends of software and potentially form the basis for eliciting improved or novel laws of software evolution. The forecasting power of the proposed model is evaluated against the actual evolution of 10 open-source projects and the achieved accuracy in the prediction of several network and software properties, which reflect the underlying system design, appears to be promising.

Index Terms—Graphs and networks, restructuring, reverse engineering, and reengineering, software architectures, object-oriented design methods

1 Introduction

Motivated by the phenomenal growth of social networks during the last decade, significant research has been performed on the study of the evolutionary trends exhibited by social networks such as Flickr and LinkedIn [46], technological networks such as those formed by web pages [19] and routers [51] as well as biological and other networks [18]. The ultimate goal in these studies is to interpret the macroscopic phenomena at the network level, relate them with the microscopic behavior of individual nodes and eventually forecast the future evolution of the corresponding networks [46].

Software systems have also been treated as networks of various forms within the software engineering community. The most common form assumes that software modules are represented as nodes while relations among them correspond to edges. Other software artifacts but also people involved in the software development process have been considered as nodes leading to different kinds of networks. Modeling software systems as networks enabled a graph-based treatment and analysis with the goal of investigating several properties, such as scale-freeness [53], [59], [67], [73], [85] and the presence of small-world phenomena [40], [77]. However, with the exception of few recent research efforts [8], [49], [93], most of the studies that employ graphs for the representation of software, focus on static snapshots of the examined systems in the sense that individual software versions have been analyzed without paying attention to the evolution of the corresponding networks.

• *The authors are with the Department of Applied Informatics, University of Macedonia, Thessaloniki 54643, Greece. E-mail: {chaikalis, achat}@uom.gr.*

Manuscript received 26 Feb. 2014; revised 18 Nov. 2014; accepted 1 Dec. 2014. Date of publication 17 Dec. 2014; date of current version 17 June 2015. Recommended for acceptance by M. Di Penta.
For information on obtaining reprints of this article, please send e-mail to: reprints.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2014.2381249

Software systems and object-oriented designs in particular, can be naturally represented as graphs. By drawing ideas from social network analysis our goal is to establish a set of techniques for studying software evolution where systems are represented as networks of interconnected nodes (classes). In particular we aim to: a) study evolutionary trends during software growth by analyzing network properties over successive past software versions, b) derive models that are capable of forecasting future software evolution in terms of network metrics, based on the observed phenomena and c) introduce domain knowledge in the construction of the corresponding models in order to improve their accuracy. The latter is of major importance when modeling a physical or technological network since the interpretation of raw data without considering domain knowledge might lead to confusing results and unreliable conclusions [87]. Whenever possible, we attempt to associate the observed trends at the network level to qualitative properties of the underlying software system.

A model that enables the prediction of the evolution of certain architectural parameters can be valuable to software maintainers. In particular, the ability to forecast the growth of classes and packages as well as the coupling among them can assist the development teams to focus on parts of the design that warrant increased attention. As it has been extensively pointed out by previous empirical studies, heavily loaded modules and excessive coupling are associated with increased fault-proneness and increased maintenance effort [8], [31], [86], [90], [95]. Moreover it has been made clear by previous research [95] that network measures can predict critical and error-prone software modules far more accurately than classic complexity metrics.

In this paper a network-based prediction model for software evolution is proposed, combining information from past data and also domain-related rules. Moreover, several processes of software growth are taken into account,

including the creation of relations among existing and new classes and the removal of edges, rather than simply relying on a model that captures how new classes are added to a system. Finally, we have observed that representing software systems as "flat" networks ignoring the existence of structural communities, significantly constrains accuracy, and thus we incorporate the notion of packages in our model. Architectural modularization when representing software systems as networks has also been considered in other research efforts [75], [91]. Obviously, a prediction model for software evolution is an ambitious goal and its accuracy is constrained by the numerous exogenous factors affecting the software development process. Nevertheless, the fact that the analysis is performed at the relatively abstract network level, enables us to obtain an insight into future trends. Moreover, as it will be shown, the obtained accuracy can be gradually improved by enhancing the model with additional parameters.

The main contribution of the proposed approach lies in the ability to estimate the network structure representing a software system instead of forecasting an individual software characteristic. Since the entire software structure can be anticipated, the benefit lies in the ability to reason about future values of several software properties emerging from the network topology.

Evolutionary trends are discussed for 10 open-source projects against which the accuracy of the proposed model is tested. The proposed study is backed up by a tool that has been developed for software evolution analysis in terms of networks. Both the results and the tool are freely available from [1].

The rest of the paper is organized as follows: Section 2 discusses related work on previous attempts that employed forecasting methods in software engineering. We also present previous works on the representation of software systems as graphs and on the evolution of software. Necessary terms are introduced in Section 3 along with details about the examined systems. The proposed approach is presented in Section 4, covering all parameters taken into account in the construction of the prediction model. Observations about small world phenomena in the examined systems and the need to evaluate the proposed model against them is discussed in Section 5. A graphical summary of the proposed model is provided in Section 6. Evaluation results are presented and discussed in Section 7 for the 10 examined open-source projects, both from the perspective of networks as well as from the perspective of software. Limitations and threats to validity are listed in Section 8 and finally, we conclude in Section 9.

2 Related Work

A large number of approaches have been developed aiming at the prediction of individual software characteristics or properties. In all cases a forecasting model is built based on the analysis of an information set (historical data, models and assumptions available at a given time) and thus every forecast is conditional on this information. All kinds of forecasting methods can be classified under the two broad categories of explanatory (or causal) and time series models. Explanatory models (including the widely used regression analysis) assume that the variable to be forecasted exhibits a causal relationship with other independent variables. Time-series forecasting treats the examined system as a black box and attempts to derive the generating process of a set of time-dependent data [15]. Quite often these approaches are enhanced by artificial intelligence (such as neural networks [79] and genetic algorithms [9]) or probabilistic techniques (such as Bayesian models [42]). Indicative approaches where a forecasting method has been employed to predict future values for a particular software characteristic are shown in Table 1. Although this list is not exhaustive it covers a large portion of software properties which have been the target of forecasting.

TABLE 1
Forecasting Methods Employed to Predict Various Software Characteristics

Characteristic	Method used
Bug Presence Error-prone components	Logistic Regression [6] Correlation Analysis [61], [96] Principal Component Analysis [3], [61], [96] Bug Caching, Historical Analysis [39] Multivariate Adaptive Regression Splines [34] Cost-benefit analysis [3]
Bug Fixing Time	Monte Carlo Simulation [92]
Bug Complexity	Clustering with k-means and k-medoids [62]
Bug Density	Correlation Analysis, Multiple Regression [60]
Cost - Effort	Expert Judgment [36] Analogy Based estimation [50], [52] Artificial Neural Networks [37], [52], [79] Bayesian Network Models [64], [76] Multiple Regression [76]
Project Scheduling	Genetic Algorithms [9]
Code clones	ARIMA Time Series [2] Dynamic Programming Matching [41] Statistical Pattern Matching [41]
Design Model Evolution	ARMA Time Series [88]
Change Proneness	Sequential pattern mining [38] Association rule mining [89]
Project Survivability	Information Theory [69]
Maintainability	Hierarchical Multidimensional Assessment [12] Polynomial Maintainability Assessment [12] Fuzzy Prototypical Knowledge Discovery [22] Bayesian Network—Naïve-Bayes Classifier [42] Regression Tree [42], [94] Multivariate Linear Regression [42], [94] Artificial Neural Network [94] Multivariate Adaptive Regression Splines [94] Non-Homogeneous Poisson Process [71]

The analysis of the evolution of software systems, as well as the attempts to facilitate the understanding of the way that systems evolve, have drawn considerable interest in the last years [58]. The significance of studying the evolutionary trends during software growth, has been highlighted quite

early in the history of software engineering as vividly captured by the laws of software evolution defined by Lehman [44]. A discussion of software evolution from several perspectives and a comparison to other kinds of evolution in various domains has been presented in the study by Godfrey and German [26]. One of the most interesting conclusions is that software evolution can offer insight into questions of both science and engineering. Different types of evolution analyses have been surveyed by Gîrba and Ducasse [24] where a set of requirements for building evolution metamodels has been proposed.

Since our model essentially encompasses the use of networks as a representation medium and as a tool to facilitate the study of software evolution, we focus here at previous research efforts related to tools for software evolution analysis as well as works that employed graphs for this purpose.

Wettel [83] and Wettel et al. [84] introduced a software visualization tool for the facilitation of monitoring software evolution that models source code packages as city districts and classes as buildings. The height of the buildings depicts the number of methods while the area depicts the numbers of attributes. In this way by examining the historical snapshots of the "software city", one can easily gain a visual, coarse-grained overview of the evolution of the software system.

Bevan et al. [7] proposed a tool called "Kenyon" to support software evolution research. Kenyon eases the extraction of data from source code repositories providing support for multiple Configuration Management Systems. The provided infrastructure enables the access and processing of collected data thus supporting several types of analyses such as the investigation of code feature evolution and history analysis in terms of graph-based software representations.

D'Ambros and Lanza [14] developed the "Churrasco" framework which enables the analysis of a source code repository located in a remote version control system (and of the accompanying bug tracking system data if available) and the modeling of software evolution. The framework provides a visualization module offering interactive visualizations concerning the evolution of dependencies among modules and size metrics, and an annotation module that supports collaborative analysis where users can share annotations on model entities.

Although not directly comparable to the prediction of future evolution that is proposed in this paper, existing research efforts in the field of change proneness prediction can also be considered as relevant, since forecasting is the goal. Gîrba et al. [25] attempt to identify software components that changed in the recent versions of software systems under the assumption that such components are more likely to change again in the near future. They also propose rules for the characterization of the evolution of class hierarchies in order to facilitate the identification of changes and change propagation among class hierarchies.

Vasa et al. [78] employed type dependency graphs to analyze consecutive releases of several object-oriented systems with the goal of understanding how changes are distributed over the classes and interfaces of systems and how these changes are affected by size, popularity and complexity of classes. They calculated 25 different metrics for each class in all consecutive project versions and compared the values between two versions to decide if the class has been changed or not. Their results indicate that as software evolves, and after a first period of intense functionality addition, the size and complexity measures of classes stabilize and do not fluctuate intensively. Another interesting outcome is that classes with high fan-in values (popular ones) are more likely to change from one version to the next, in other words the increased size, popularity and complexity of a class affect its change-proneness.

The proposed use of networks/graphs for software evolution analysis is based on the fact that several artifacts of the software development process have long been modeled as graphs. Such graphs have been employed to represent, for example, data or control dependencies between statements, relations between software components, links between project activities and priorities between requirements [23]. Many researchers have also studied the properties of the corresponding networks such as the presence of power laws in the distributions of different software components [13], [20], [53], [59], [67], [72], [74], [85] the existence of small-world phenomena [40], [77] and the definition and identification of network motifs [55], [77]. Taube-Schock et al. [73] studied connectivity in 97 open source systems and concluded that scale-free structure in the source code translates directly to high coupling and therefore claim that high coupling cannot be eliminated from software design.

Furthermore, several approaches have tried to explore the potential of network properties, such as node degree and centrality, to act as software quality indicators or bug predictors. Pinzger et al. [66] represented the interconnection of developers with software modules in a network structure, and found a correlation between the centrality of software modules in the network and the error proneness of these modules. A similar work has been carried out by Meneely et al. [57]. Zimmermann and Nagappan [95] modelled the interactions among binaries of the Windows Server 2003 as a binary dependency network and investigated the correlation between several network measures (especially centralities) and the number of defects. Their results highlight the fact that network measures are much better predictors of critical binaries than the traditional complexity metrics. They also claim that binary dependency network metrics can predict the number of defects. Turnu et al. [75] propose the fractal dimension network metric as a software complexity indicator. They also provide evidence of correlation between the fractal dimension metric and the number of defects. Zanetti and Schweitzer [91] proposed a metric that quantifies the modularity of a system by using its representation as a software network.

Paymal et al. [63] have investigated the changes to graph vertex properties such as degree, betweenness centrality, clustering coefficient and measured the extent of disruption after perfective maintenance in the evolution of JHotDraw application. Wang et al. [80] studied the evolution of the call graphs corresponding to 223 consecutive versions of the Linux kernel, employing several graph properties as metrics, observing that the network growth relies heavily on preferential attachment. A network representation of the Linux operating system has also been studied by Fortuna

et al. [20], who analyzed dependencies among packages for the first 10 releases of Debian. According to their findings, the system exhibits high modularity which increases with the passage of versions, although not with a constant rate. The authors claim that the increased modularity, albeit not totally preventing incompatibilities among modules, helps to avoid conflicts between software sub-components and eases the installation of autonomous modules.

A more systematic study of the evolution of networks representing software systems as well as the introduction of models that govern their evolution has also been attempted by a limited number of researchers. Li et al. [49] proposed a model according to which, software networks grow not only by individual node additions, but also with multiple node attachments in the form of complete modules. The model starts with the insertion of single nodes in the existing network, but after some initial steps the number of nodes added in each step is increasing. At first, the nodes to be inserted form a new sub-network S′ according to the Preferential Attachment (PA) model. Then the entire sub-network S′ is attached to the existing network according to a set of defined rules that govern the edge creation and the target node selection.

Bhattacharya et al. [8] capture software structure by means of networks, at two levels, namely the source code and the developer collaboration level. A set of graph metrics such as number of nodes and edges, average degree, clustering coefficient and edit distance, is employed to study the evolution over the entire lifespan of 11 open-source projects. The employed metrics have been shown to be valid predictors of bug severity, high-maintenance software modules and failure-prone releases.

Our study differs from the aforementioned efforts in that the proposed approach aims at predicting the future evolution of the entire network topology representing an object-oriented system. Moreover, the proposed prediction model considers a multitude of parameters such as generic growth models (e.g., preferential attachment), past evolution data as well as domain knowledge for object-oriented design. In addition, system structure in the form of packages is also considered in the proposed model.

One representative case where the consideration of additional parameters appears to improve the forecasting power is the work by Zheng et al. [93] who represented the Gentoo Linux open-source operating system as a complex network with software packages as the nodes and inter-package dependencies as the edges in an attempt to study its evolution by means of networks. Their empirical results indicated that existing network models are not capable of predicting the actual evolution of the software systems and therefore they propose an evolution model that considers node age in parallel with node degree, called Degree and Age Dependent Adjustable Evolution (DAAE) model. The introduction of age seems to improve the forecasting power of their model, which however has not been evaluated thoroughly.

The conclusions of this approach are in line with our observations that the Preferential Attachment model by itself is not capable of forecasting the evolution of software systems due to their inherent special characteristics. However, apart from taking into account the node age, we also consider specific domain rules that govern the development of software systems. Furthermore we model the creation of edges between existing nodes, between existing and new nodes as well as the removal of existing edges, issues that have not been taken into account in previous models but which fundamentally affect the topology of the resulting software networks. Finally, the evaluation is performed on 10 open-source systems (models proposed by Zheng et al. [93] and Li et al. [49] are evaluated on a single system), which enables the generalization, up to a certain degree, of the observed evolutionary trends in software evolution.

3 Case Study Design

3.1 Terminology

As already mentioned we treat object-oriented systems as networks of interconnected classes. Nodes correspond to classes, including abstract and concrete ones. In particular we model the network of classes according to the Law of Demeter (LoD) [48]. An edge linking two nodes (source and target) indicates that between the corresponding classes there is an 'allowed' (according to the Law of Demeter) relationship. In the LoD context, a relationship is allowed in case the source class contains 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. The resulting graph $G < V, E >$, where V and E is the set of nodes (vertices) and edges, respectively, is directed.

During the evolution of an object-oriented design over a number of versions, new nodes (classes) might be added in any version. All other classes are considered as existing nodes. Given these two types of nodes, the following types of edges have to be taken into account in the development of a prediction model:

- edges leaving from new and reaching existing nodes
- edges leaving from new and reaching new nodes
- edges leaving from existing and reaching any other node.

3.2 Examined Systems

Evolutionary trends have been investigated for 10 open-source systems, which are also the systems against which the proposed prediction model has been evaluated. The systems have been selected according to the following criteria:

- they should be open-source projects since the source code has to be analyzed in order to extract its network representation
- they should be written in Java since our tool [1] is currently capable of analyzing Java source code.
- they should have varying sizes to test more effectively the scalability of the model. (The selected projects' size ranges from 55 to 3201 classes)
- a good number of versions should be available, in order to allow an adequate capturing of the project's historical evolution.

A brief description of each project is provided in Table 2. The size characteristics of the networks corresponding to the first and last version of each examined project are shown in Table 3 along with the number of successive versions that have been used in the analysis.

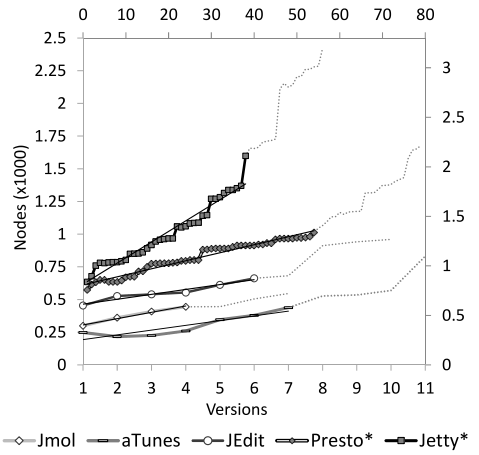
TABLE 2
Projects under Study

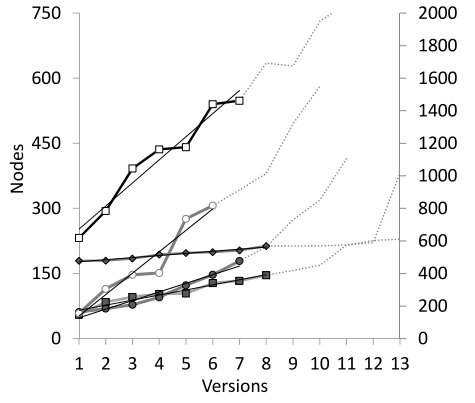
aTunes	Audio player and music library organizer that supports many types of audio formats [4]
FreeCol	Turn-based strategy game similar to Civilization with graphical user interface [21]
JDeodorant	Eclipse plug-in for the detection and elimination of design flaws by means of refactoring [29].
JEdit	Cross-platform text editor, which can be customized by plugins [30]
JFreeChart	Chart creation library with an extensive variety of supported charts [33]
Jmol	Viewer that visualizes chemical structures in 3D [35]
Weka	Machine learning software suite that contains a collection of visualization tools and algorithms for data analysis and predictive modeling. [82]
HFSExplorer	Application that reads and manipulates Mac-formatted hard disks and disk images. [27]
Presto	Facebook's distributed SQL query engine for running interactive queries against big data sources. [68]
Jetty	Web Server and Servlet Engine with support for SPDY, WebSocket, OSGi and JNDI technologies. [32]

The number of nodes (classes) for each examined version and for all analyzed projects is graphically depicted in Fig. 1. The evolution of each project is split in two periods: 1) the training period on which the evolution has been monitored to configure the model parameters (continuous line) and 2) the testing period against which the forecasting power of the model was tested (dotted line). A linear trendline is fitted on the evolution of each project during the training period. The goodness of fit (R^2) for each project is shown below the charts. As it can be observed the nodes increase in an almost linear fashion in all cases (the goodness of fit R^2 to a linear function ranges from 0.83 to 0.98). Thus, to avoid unnecessary complexity to the proposed model, we obtain the number of new nodes for each forthcoming software version as the mean value of new nodes per version, of all past software versions. However, it should be noted, that the proposed model does not necessarily rely on a linear fit on the node evolution to estimate the number of nodes that should be added in each version. If historical data can be better captured by a different type of growth model (e.g., logarithmic or exponential), the number of nodes can be estimated accordingly.

TABLE 3
Project Characteristics

#	Name	Examined Versions	Nodes		Edges	
			First	Last	First	Last
1	aTunes	11	249	832	570	1,979
2	FreeCol	10	232	772	732	3,975
3	JDeodorant	13	55	229	184	780
4	JEdit	12	455	960	1,035	2,278
5	JFreeChart	13	478	1,018	1,575	2,892
6	Jmol	8	316	547	615	1,130
7	Weka	13	156	1,550	599	5,884
8	HFS Explorer	11	61	414	125	1,233
9	Presto	79	758	2,219	2,604	7,595
10	Jetty	58	838	3,201	1,652	7,952





Project	Tunes	FrCol	JDeo	JEdit	JFC	Jmol	Weka	HFS	Presto	Jetty
R^2	0.83	0.95	0.95	0.95	0.96	0.98	0.93	0.95	0.96	0.94

Fig. 1. Evolution of node count and R^2 of the corresponding trendlines.

In the following sections all aspects of the proposed model will be presented, attempting to justify each decision in relation to the historical trends which are present in the evolution of the examined systems as well as the corresponding phenomena which have been analyzed in other domains. An overview of the entire model is depicted graphically in Section 6. Beyond the parameters which are dictated by the proposed model, we leave for each choice that has to be taken, a certain degree of randomness, since we acknowledge the fact that there are dimensions in the software evolution process which are not strictly following rules or past distributions.

The reason for which we propose a graph based model for the prediction of software evolution is that numerous factors come into play which cannot be accounted for by a simple regression-based model. To illustrate the inherent limitation of regression as a means of forecasting the evolution of software architecture parameters, such as the afferent coupling of a class, we illustrate in Fig. 2a regression-based forecast of the in-degree.

In particular, the evolution of the in-degree for classes that exhibited the largest variation in their in-degree between the first and last examined version, is shown. The

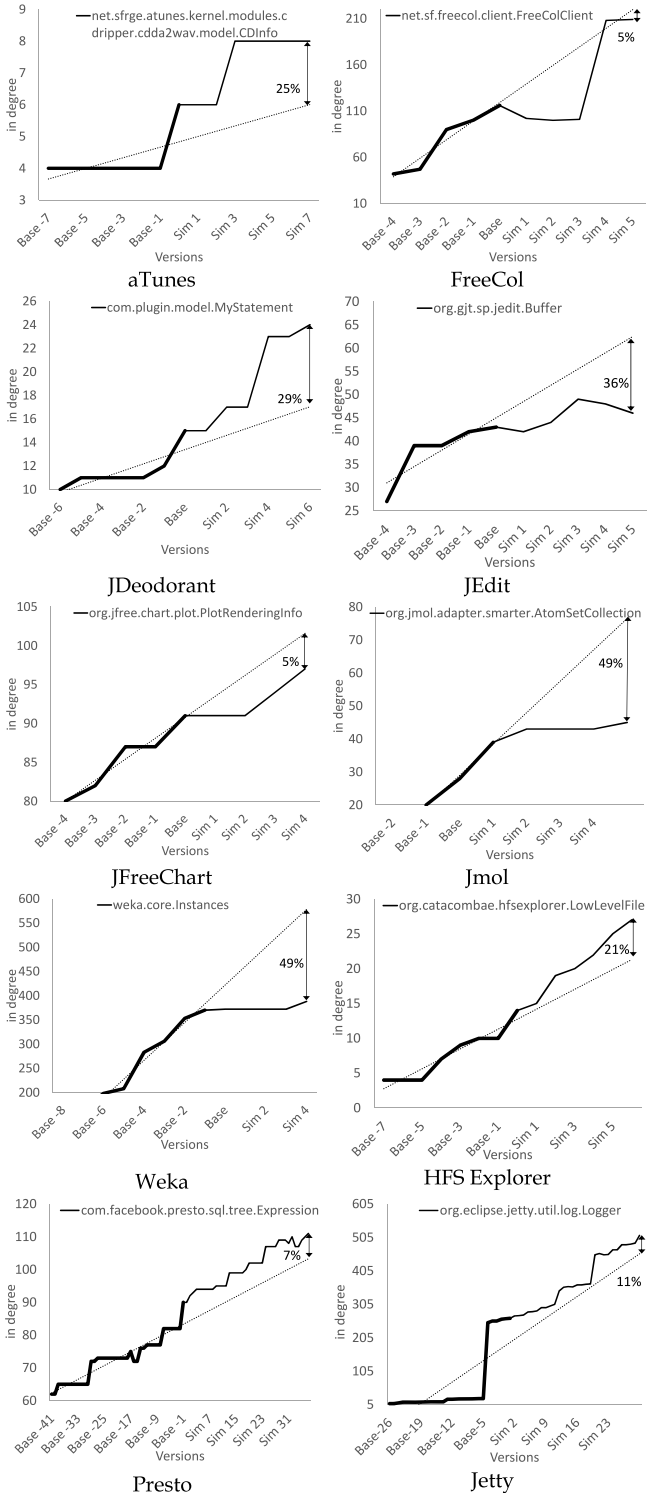


Fig. 2. Regression based forecast of the in-degree for the classes that exhibited the largest variation in their in-degree between the first and last examined version, for each of the examined projects. The thick line corresponds to the training period. The dotted line corresponds to the linear regression on the training period along with the projection up to the final version.

plot includes also the trendline corresponding to a linear function (providing the best goodness of fit), fitted to the data of the versions that form the training set. The size of the training set is set to 50-60 percent of the total number of versions of each project. The limited ability of the regression model to correctly predict future changes is evident in most cases. The predicted in-degree for the last version differs from the actual in-degree and it is apparent that regression cannot account for the variation in the in-degree that takes place during the evolution of a class.

4 Model Description

A network topology evolves over time by two mechanisms: addition/deletion of nodes and addition/deletion of edges. All other network properties emerge from the resulting network topology. What we have attempted in our approach is to model the node arrival process, the edge creation process (selection of source and target nodes) and the edge removal process (which might also result in node removal).

There is a plethora of properties which can be measured during the evolution of a network, related to all perspectives from which a graph can be studied such as size, connectivity, cycles, coloring, cliques, shortest paths, centralities, etc. From all available graph properties, our survey of numerous models that have been examined in the context of other disciplines revealed that the node degree is the primary measure of interest and common in all studies. For this reason we have incorporated the study of degree distributions in our model as well.

However, as it will be made clear in the evaluation, models which only consider mechanisms that explain changes in the node degree do not manage to predict accurately the network topology. To improve the accuracy of the edge and node creation processes we have incorporated the effect of source/destination node age because we have observed that this property exhibited regular patterns during the evolution of the examined systems whereas other properties did not exhibit any noteworthy motifs.

The model has been further refined with two other functions related to the software nature of the examined networks: a) the duplication mechanism to reflect the package-level structure of each system and b) the restrictions on node behavior dictated by domain rules.

4.1 Preferential Attachment and Duplication

One of the most striking observations that has been repeatedly made for several technological, biological and social networks, despite their inherent differences, is that very often they all exhibit a common pattern in their degree distribution. The pattern consists in a degree distribution that is heavy tailed and usually follows a power-law, a behavior that has been considered as a strong indication of a scale-free network [5], [73], i.e., a network that is self-similar at all scales. In other words, the fraction of nodes $P(k)$ having k edges decays following a power law $P(k) \sim \mathrm{k}^{-a}$, with the value of parameter a depending on the nature of the application domain. A different interpretation of the above phenomenon states that: *The probability that a node n has degree d(n) is proportional to* $d(n)^{-}\alpha$:

$$P(d(n)) = Cd(n)^{-\alpha} \tag{1}$$

where C is a normalization constant and $a > 1$.

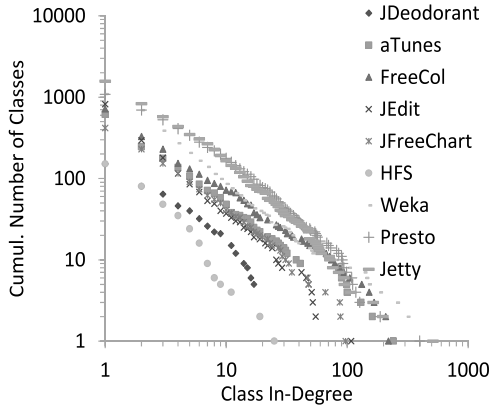
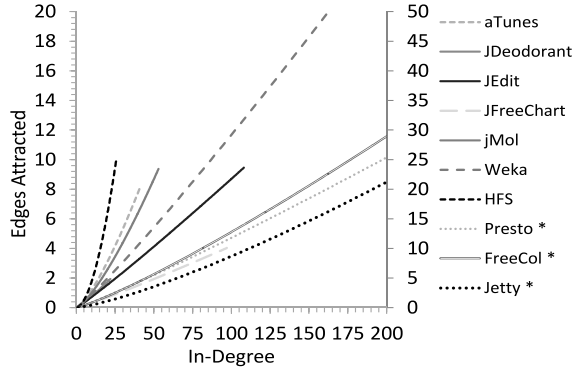


Fig. 3. In-degree distribution for the examined systems.



* Projects Presto, Freecol & Jetty are displayed against secondary y- axis

Fig. 4. Node in-degree versus number of edges attracted.

If we take the logarithms on both sides of (1), we end up with the following equation:

$$\log(P(d(n))) = -a\,\log(d(n)) + \log(C) \tag{2}$$

which manifests itself as a straight line in log-log plots of d versus $P(d)$, with $-a$ being the negative slope of the line.

To investigate whether a power law is also valid for object-oriented software systems we plot in Fig. 3 the in-degree distribution of the classes for the last version of each examined system. In particular, the figure shows the cumulative number of classes vs. the class in-degree on a log-log plot. The almost straight form of these distributions confirms the presence of a heavy tailed degree distribution. As it would be reasonable to expect, very few classes have a large in-degree (providing services to a large number of dependent clients) while quite many classes are accessed by a limited number of other modules leading to a low in-degree. Similar conclusions regarding the existence of power laws in large software systems have been drawn by other studies [13], [20], [53], [73], [85].

The generation of scale-free properties and power law distributions is usually attributed to the presence of Preferential Attachment, which postulates that when new nodes are added to an existing network forming edges to existing nodes, the number of edges attracted by each target node is proportional to the target's in-degree [5]. In an object-oriented setting, this evolution model, commonly known as the "rich-get-richer" rule, implies that classes having already a central role in the system (i.e., are a form of "God" classes providing services to numerous clients [70]) act as attractors for new classes that join the system.

Moreover, it has been observed that this phenomenon becomes more intense as software evolves. With the passage of versions a larger percentage of new classes are linked to higher in-degree classes. Stated in a different way, "*important nodes* (in software systems) *stay important*", a fact which is considered as a sign of stability in the design in the work of Paymal et al. [63]. To validate empirically whether Preferential Attachment actually holds for the systems under study, we illustrate in Fig. 4 the number of edges attracted by a node versus the in-degree of this node. The number of edges attracted by a node of a particular degree has been calculated as follows:

During the evolution of a system, there are numerous nodes having a particular in-degree at each time point. Moreover, a node having a particular in-degree value at a certain time point t might have a different in-degree at a later time point. The average number of edges attracted by nodes having in-degree k should be calculated considering all versions [46]. The corresponding formula is

$$AvgNumOfEdges = \frac{\sum_{forAllVersions} |\{e(u,v) : d^-(v) = k\}|}{\sum_{forAllVersions} |\{v : d^-(v) = k\}|} \tag{3}$$

where:

$d^-(v)$ is the in-degree of node v

$e(u, v)$ is an edge from node u to node v

The numerator represents the number of edges towards nodes of in-degree k and the denominator is the number of nodes having in-degree k.

As it can be observed from Fig. 4 most new edges are attached to classes that have a large in-degree, while low in-degree classes appear to attract a limited number of new edges. Thus, it can be concluded that the preferential attachment mechanism is strongly present in the evolution of the examined software systems.

Based on these observations we adopt the Preferential Attachment model as a core element in the construction of our network prediction model. Once a new node is added in a version and the number of outgoing edges that will be formed is determined (as it will be explained next), the target nodes are selected with a probability that is proportional to the class in-degree. According to Barabási and Albert [5] this probability P for node v depends on the in-degree $d^-(v)$ of the node and is equal to

$$P(v) = \frac{d^-(v)}{\sum_{i=1}^{|V|} d^-(i)}, \tag{4}$$

where $|V|$ corresponds to the cardinality of the vertex set of the graph.

However, the consideration of the preferential attachment mechanism is not sufficient for modeling the evolution of software systems. In evolution models of other domains (biological, social or technological), most systems are represented as networks in a rather 'flat' way, that is, all new nodes are assumed to belong to the same community neglecting distinct sub-communities which might exist.

However, in software systems, structure in terms of packages or namespaces heavily influences the way that a network evolves and thus should not be ignored [49]. According to our experiments applying the preferential attachment model considering all system classes as a single community without structure, limits significantly the explanatory power of the model. To confront this issue we adopted, on top of the preferential attachment, the so called Duplication Model according to which the behavior of a new node that is introduced to a network copies the behavior of a previously introduced node [10], [11], [40], [43], [77].

Following this model we construct for each analyzed system a data structure holding all previous node behaviors. By the term 'behavior' we mean for each 'new' node that has been introduced in the past, the software packages to which it has created edges, the number of edges that have been created to each package and in which package the node itself has been added. Consequently, when modeling future evolution, we select a previous behavior and copy it.

What remains open is the specific target class of the selected package to which the new class should be attached. For this parameter we rely on the preferential attachment model. As a result we employ a combination of these two models: we rely on the Duplication model for determining the coarse-grained aspects of package selection (and as a consequence we introduce domain knowledge into the evolution model) and for the more fine-grained process of class selection we employ the preferential attachment model (which in turn ensures the preservation of the power law phenomena).

4.2 Modeling Node Activity

Obviously, an object-oriented system does not evolve simply by adding new classes. The already existing classes also form new relations among them in order to access new functionality or data. Neglecting the evolution of edges emanating from existing nodes would lead to a highly inaccurate model. To incorporate this aspect in our model, three issues should be addressed

a) Determination of the number of edges to be created
b) Selection of the source node for each new edge
c) Selection of the target node for each new edge.

In order to determine the number of new edges that should be formed by existing nodes, we rely on past data and particularly on the number of new edges created by existing nodes in all previous versions. In Fig. 5 the cumulative frequency of the number of new edges (emanating from existing classes) is shown for all examined systems. As an example, point *A* in the plot indicates that (for project FreeCol), in 60 percent of the cases, up to 200 new edges have been formed by existing nodes. During the simulation of future evolution the number of new edges to be formed by existing nodes is sampled from this distribution through inverse transform sampling [16].

To perform inverse transform sampling from a cumulative distribution function (CDF) the first step is to fit an appropriate regression function f on the given data points and obtain its inverse function f^{-1}. Next, a random y value in the range [0, 1) is chosen and applied to the inverse function yielding the corresponding x value. In our case, the obtained x value indicates how many edges should be formed by existing classes for a particular simulation step.

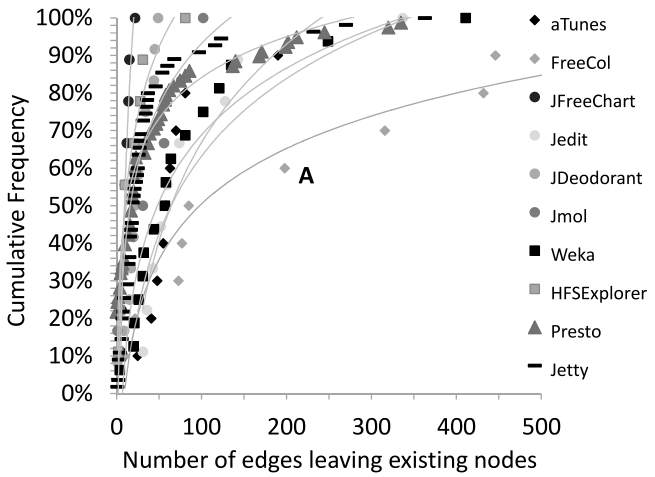


Fig. 5. Distribution of the number of new edges leaving existing nodes.

The source node for each new edge is selected by employing the previously mentioned Duplication Model since even for edges created by existing nodes, the system's structure in terms of packages should be respected. As in the previous section, the duplication model determines also the target packages to which the source node will form edges. The target node for each new edge is selected according to the Preferential Attachment model as previously, since large classes in terms of incoming relations act as attractors also in this case.

When new classes are introduced to an object-oriented system they form relationships to existing (older) classes but very often these new classes also collaborate with each other. This is reasonable, since classes are not added as individual modules but as pieces of code offering certain functionality and thus they are exchanging messages among them. Quite often, the new classes are added in the form of an entire new package with a complete architecture as it has also been observed in the work of Li et al. [49]. For these reasons, a prediction model should also consider the edges that are formed between new nodes, i.e., leave a new (source) class and reach a new (target) class. For the determination of the number of edges between new nodes, we follow the already explained strategy by sampling from the distribution of past data concerning edge creation among new classes in each version.

4.3 Effect of Class Age

The study of software evolution in terms of the underlying networks, over several versions and different projects, reveals another interesting feature regarding node activity: The majority of edges are departing from "younger" nodes, with their age measured in number of versions. This remark is in agreement with similar observations that have been made for the evolution of software systems in other studies [8], [93]. The creation of outgoing edges is equivalent to the access of functionality and data offered by other classes, in order to accomplish the functions that the source class should deliver. Evidence from the systems

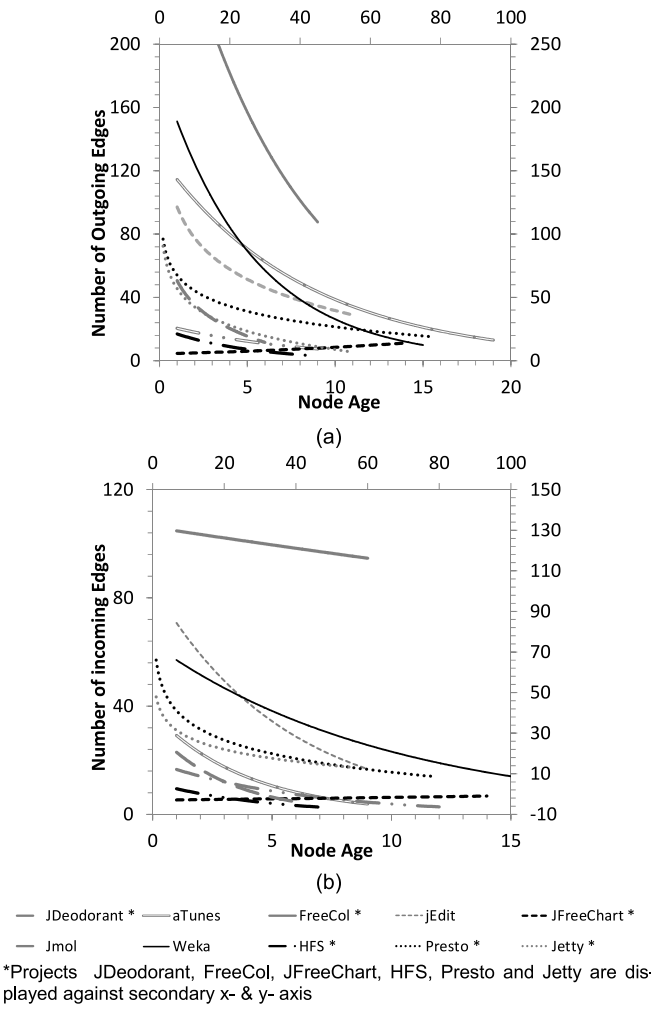


Fig. 6. (a) Total number of outgoing edges versus node age and (b) Total number of incoming edges versus node age.

that have been examined clearly support this viewpoint: Fig. 6a displays, for all examined systems, the total number of new outgoing edges created from nodes which at any time point of their history had the corresponding age (the lines correspond to exponential fits on the actual data). It becomes clear that among the created edges, most edges are departing from "young" nodes. This fact is taken into account in the prediction model, by incorporating the node age in the Duplication Model applied for the selection of the source nodes. As a result when a node has to be selected as the source of a new edge, the model does not simply rely on past node behaviors but restricts the selection among nodes with the required age, which is obtained by inverse transform sampling.

In a similar manner the age of a class affects its probability of being the target node for a new edge. As already explained this is mainly determined by the preferential attachment model, according to which classes with a large in-degree act as 'attractors' for new edges. However, we have observed that the raw application of the Barabási and Albert model [5] leads to an excessive loading of large classes with additional incoming edges. To mitigate this issue and extract a more accurate model we do take into account the age of the target node as well. Fig. 6b illustrates the total number of incoming edges versus the age of the target

TABLE 4
Number of Added and Removed Edges for the Examined Projects and Versions

Project	Edges Added	Edges Removed
aTunes	5,621	3,344
FreeCol	4,834	1,591
JDeodorant	765	169
JEdit	2,590	1,347
JFreeChart	522	134
Jmol	1,406	888
Weka	9,727	2,735
HFS	1,470	361
Presto	15,346	7,190
Jetty	12,570	5,216

node, for all past data of the examined projects. Among the created edges, most incoming edges reach "young" nodes, but this tendency is less intense than for outgoing edges. It should be noted that this observation is not in contrast to the Preferential Attachment mechanism. The large number of incoming edges attracted by "young" nodes (which usually have low in-degree) occurs because we take into account edges in the entire history of a project. By employing inverse transform sampling on the corresponding cumulative distribution, we obtain for each new edge to be created the desired age for the target node. Consequently, in the proposed software evolution model both the appropriate in-degree (according to the preferential attachment model) as well as the appropriate age (according to inverse transform sampling) of the target node are considered.

4.4 Edge Removal

Our empirical investigation indicated that during the evolution of the examined systems, apart from the introduction of new nodes and the addition of new edges, a significant number of edges have also been removed. Table 4 shows the number of edges that have been added and removed during the evolution of each examined system (over the versions that have been taken into account).

It is apparent that the removal of edges cannot be neglected during the construction of a software evolution model. On the contrary, the number of removed nodes is insignificant (for all projects the percentage of deleted nodes versus the number of nodes in the last version ranges around 2-3 percent).

In order to simulate an edge removal one should decide on the source and destination node of the edge to be removed. In the context of our model, the parameters that could have been considered in such a selection are the nodes' age and degree. The history of all examined systems revealed that there is no clear relationship between the age of the source and destination node, whenever an edge is removed. On the other hand, there appears to exist a relatively strong relation between the out-degree of the source node (and the in-degree of the destination node, respectively) and the number of removed edges. Fig. 7 illustrates the number of deleted edges versus the (in/out) degree of the deleted edge's source node and target nodes, for all examined software projects.

It appears that as the out-degree of a node becomes higher (possibly implying the accumulation of functionality

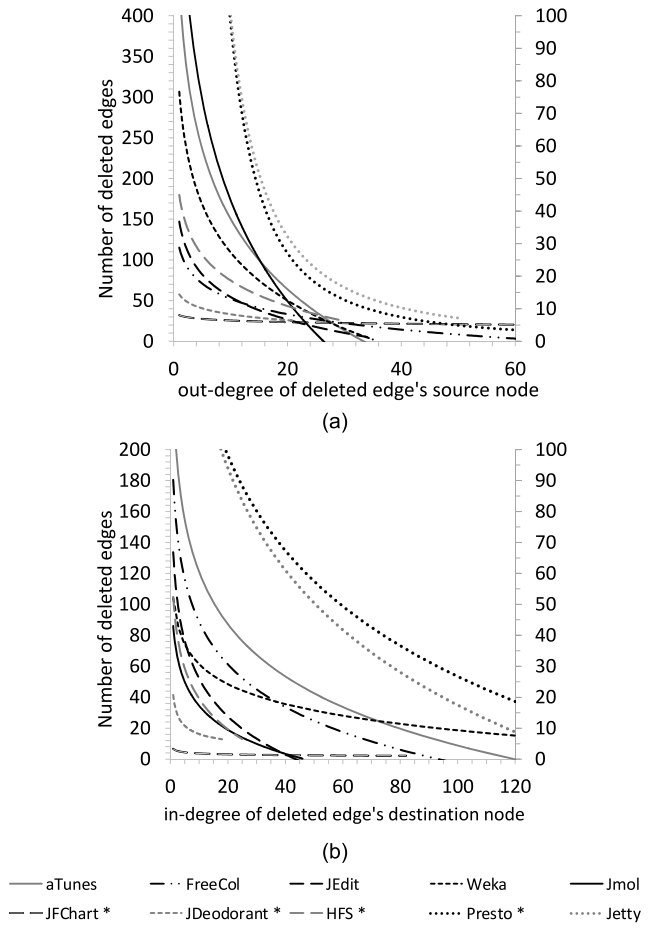


Fig. 7. Number of deleted edges versus (a) out-degree of deleted edge's source node and (b) in-degree of deleted edge's destination node. Each curve corresponds to an examined project.

or data in the corresponding class), fewer edges are removed from that class (Fig. 7a). The same holds for the relation between the number of deleted edges and the in-degree of the destination class (Fig. 7b): the largest percentage of removed edges seem to reach nodes that have a relatively low in-degree. Once again, this past information is taken into account in the proposed model, by converting these distributions into cumulative form and then by applying the aforementioned inverse transform sampling methodology to extract for each edge to be deleted the sought degree of the destination and target class. Next, an edge is removed for a pair of nodes that satisfy the required degrees. In case the removal of an edge leads to an unconnected node (orphan node), the corresponding node is also removed. To determine the number of edges to be removed in each simulated version, as already applied in other aspects of the model, we sample from the cumulative distribution of past edge removals.

4.5 Introduction of Domain Knowledge

Willinger et al. [87] vividly showed that interpreting phenomena at the network level, such as the presence of scale-free properties, without proper handling of domain-specific issues might be a cause for enormous confusion. For example they have debunked the myth of scale-free internet formed by interconnected routers. Numerous researchers observed an apparently striking common characteristic, according to which their node connectivities follow a scale-free power-law distribution [5]. However, this conclusion stems from "unhealthy" data: For the case of router networks, neglecting domain specific issues, such as IP aliases and the inherent inability of the traceroute tool to reveal the actual node degree might generate misleading results. As a result, the excitement generated by straight-line behavior in log-log plots of degree vs. frequency as evidence for power-law phenomena might be based on unreliable data sets. The authors suggest *to rely on domain knowledge and exploit the details that matter when dealing with highly engineered systems* [87].

Obviously, object-oriented designs are also engineered systems where particular tradeoffs have been considered during their construction. These tradeoffs are exactly what differentiates them from social networks. As an example, in a social setting, there is no cost in creating a friendship relation to another person and in most cases, the more friend connections the better. On the other hand, in an object-oriented system, forming a relation between two classes increases coupling and the associated impact on the architecture is usually taken into account.

Consequently, to improve the accuracy of the prediction models for future software evolution trends, domain knowledge should be incorporated. Here, we do not aim to provide a full coverage of rules that govern the evolution of software. This would be a great challenge given that different development teams set different priorities and follow distinct principles. However, it will be shown that even by considering one common practice in OO design can largely improve the forecasting accuracy.

The Dependency Inversion Principle [54] (and in part the Open Closed Principle) in object-oriented design states that details in the design (i.e., concrete classes containing implemented methods) should depend on abstractions (such as abstract classes and interfaces) rather than having abstractions depend upon details. The second formulation of the Dependency Inversion Principle states this explicitly, as "*Abstractions should not depend upon details*" [54]. This well-known principle promotes the separation of interface and implementation, by having concrete subclasses extend the abstractions, achieving increased flexibility and reusability.

Moreover, abstractions are considered stable (especially in the case of pure abstract classes or interfaces) in the sense that they are depended upon by many other classes [54] and thus there is a good reason for keeping them unmodified. Under this perspective, designers should strive to preserve the stability of abstract classes so that changes in them do not ripple up to their clients causing them to change as well. In terms of relations among classes, the conformance to the stability principle dictates that abstract classes should not often form relations to other concrete modules causing a change in their interface. This is not to claim that abstract classes are not associated to other classes but rather to emphasize that once an abstraction has been introduced to the system, the outgoing relations to other modules should not change drastically in later versions. In the context of the proposed model this design rule (Rule 1) implies that

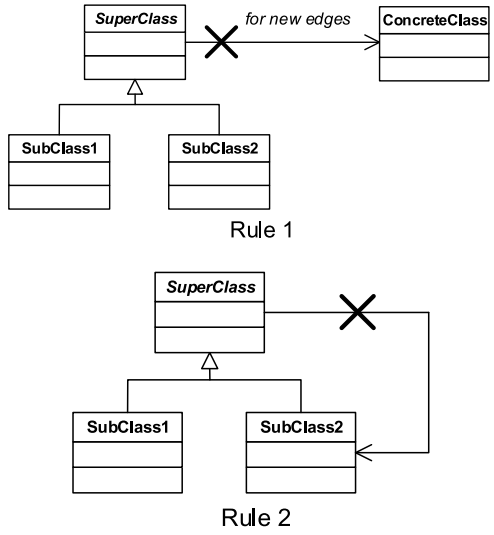


Fig. 8. Graphical depiction of applied domain rules.

creating new edges that leave abstract classes (and reach other system classes) should be avoided. The same holds for Java interfaces whose stability is even more important (an outgoing edge from an interface would occur by having another class as parameter in one of its methods).

An even more clear violation of this principle occurs whenever a superclass contains a reference to any of its subclasses. According to Riel's design heuristics, "*base classes should not know anything about their derived classes*" [70] as this would force an abstraction to become dependent upon a concrete implementation (detail) canceling out the benefit of inheritance. In the context of the proposed model this rule (Rule 2) implies that new edges leaving abstract classes and reaching their subclasses should not be allowed.

The aforementioned two rules, as applied in the proposed model, are depicted graphically in Fig. 8. The model rejects edges that correspond to the marked cases.

Practically, during the construction of networks for future software versions it is being checked whether the following domain rules are satisfied. Whenever an edge that violates these rules is created, it is being rejected and another source or destination node is being sought.

The validity of these particular domain rules can be tested by examining the occurrences in which the rules have been violated throughout the history of the examined projects. We recorded separately the number of times that a superclass created an edge to another system class and the number of times when a superclass created an edge to one of its subclasses. In particular, for the first rule, we calculated the percentage of edges throughout the history of each project that emanate from super classes and end up in another system class (not subclasses). For the second rule, we calculated the percentage of edges that link a superclass to one of its subclasses. The results are summarized in Table 5.

As it can be observed, although the violation of rule 1 is not common, the percentage is not negligible. As stated in Section 3.2, the proposed prediction model allows, via a model parameter, for a certain degree of randomness in every decision that has to be made. Consequently, a certain

TABLE 5
Percentage of Actual Rule Violations in the History of Examined Projects

#	Name	Rule 1 violations	Rule 2 violations
1	aTunes	10%	0%
2	FreeCol	6%	0%
3	JDeodorant	5%	0%
4	JEdit	8%	0%
5	JFreeChart	10%	0%
6	Jmol	15%	1%
7	Weka	6%	0%
8	HFS Explorer	20%	2%
9	Presto	8%	0.1%
10	Jetty	10%	0.1%

percentage of rule violations can be permitted. By examining these cases in detail, we have found out that such new edges (forming new links between an abstract superclass and another system class) is always the consequence of enhancing the public interface of the superclass. As an example, the introduction of a new method signature, containing a parameter of a system class type, in the list of methods of an interface, leads to an admissible violation of rule 1. On the contrary there are almost no violations of rule 2 as it would be a major design flaw to make a superclass dependent upon its descendants.

The effect of the consideration of these rules on the accuracy of the proposed model will be demonstrated in Section 7 where the results are discussed.

5 Small World Phenomena

Social psychologist Stanley Milgram conducted in the 1960s a famous experiment which suggested that the entire human society is characterized by relatively short paths among its members. Popularly known as six degrees of separation this experiment concluded that any two people on this planet can be linked via an average number of six intermediate acquaintances [18], [40]. A similar property has been observed in several technological and social networks [81]. A network is characterized by the *small-world phenomenon* if any two nodes have a high probability of being associated through a short path of intermediate nodes [18].

The presence of the small-world phenomenon implies that the diameter of the corresponding graph is relatively small. According to Easley and Kleinberg [18] the small-world phenomenon can be attributed to the existence of homophily (the tendency of nodes to connect to similar nodes) and the presence of weak ties (edges that link distant nodes). In object-oriented systems both of these properties are reasonable to expect. Up to a certain degree, classes tend to form relationships with other classes that have conceptual similarities (such as classes residing in the same package) and at the same time classes often create associations to distant classes residing in other packages. Therefore, we should expect that the majority of classes in networks representing object-oriented systems are connected by paths of a relatively small length.

To investigate the presence of the small world phenomenon we employ hop plots, depicting graphically the number

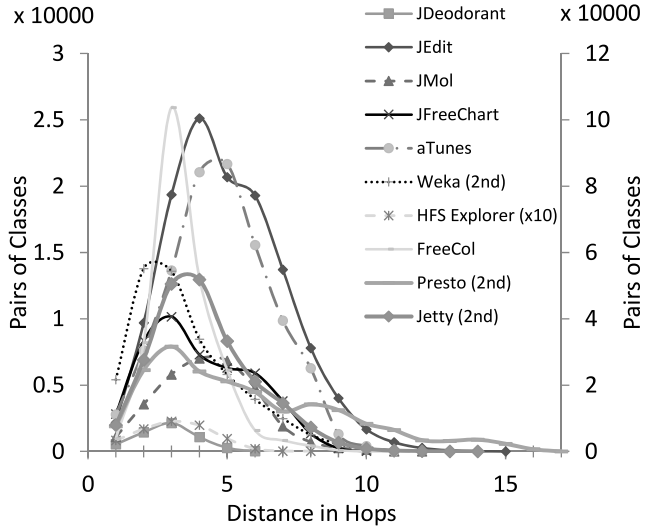


Fig. 9. Hop Plots for the last version of all examined systems (curves are smooth for aesthetic reasons: distance can assume only integer values).

of class pairs which are h hops apart. Fig. 9 illustrates the hop plot for the last version of all examined systems. The most striking observation is the spike around 3 and 4 hops, indicating that in almost all systems, the majority of class pairs have a distance of 3 or 4 hops. Moreover, even for the systems with a larger diameter, very few classes are more than 9 hops apart. This initial evidence implies the presence of a sort of small-world phenomenon in the networks of classes. However, its intensity varies from one project to the other. It should be noted that if the main goal is to test whether a network exhibits small-world phenomena, a more systematic approach could be employed, such as the comparison of topological parameters (e.g., clustering coefficient) to a null model [28].

In order to investigate whether the decisions that have been incorporated in the proposed model (such as the preferential attachment and duplication models, the sampling of past distributions and the introduction of domain knowledge) preserve the validity of the small-world phenomenon, we consider it important to assess the accuracy of the proposed model employing the aforementioned hop-plots. We believe that these diagrams reflect in a representative manner the structure of the underlying graph and as a result can reveal whether the forecasting power of the proposed model is high or low. Therefore we have used hop plots that compare the predicted network structure to that of the actual target network in the evaluation section (Section 7).

6 Model Overview

Since the proposed model encompasses a number of phases and parameters, a graphical overview is provided in Fig. 10. First, the project versions under investigation are parsed and mapped to their corresponding network representation (step 1) and these representations are analyzed to extract the aforementioned past distributions (step 2). Next, a sequence of steps is repeated for all versions that have to be simulated, in order to perform changes on the virtually evolved network. All actions that have to be performed are depicted in the shown steps along with the parameters that have to be considered.

All aspects of the proposed model have been implemented as an Eclipse plugin. The plugin can be downloaded from [1]. The webpage provides also access to the network data for all projects examined in this paper.

7 Evaluation

Since the proposed model aims at predicting future trends in software evolution, in terms of its network representation, evaluation consists in determining its forecasting power. In order to check the efficiency of the proposed model we employ the approach that has been used for similar purposes in the domain of social networks. A set of past version data (in our case 50-60 percent of the available software versions for each project) is employed as training dataset in order to obtain the distributions of network properties that dictate the model parameters.

The rest of the versions are used as test dataset and eventually we perform the evaluation along two axes:

Network perspective. We test network properties derived from the application of the proposed model, against the values of the corresponding properties for the final available software version, and

Software perspective. In order to demonstrate the potential for practical exploitation of such a prediction model, we investigate the forecasting power when predicting the evolution of class fan-in as well as the afferent and efferent package coupling.

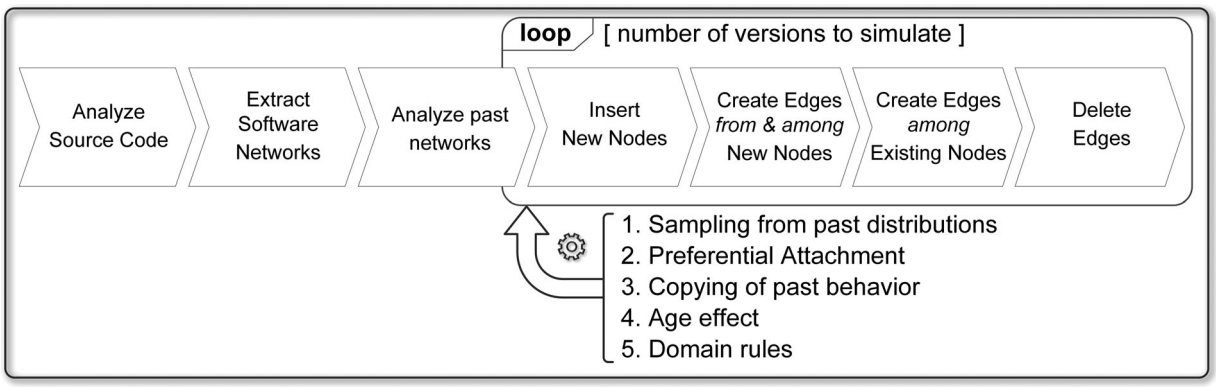


Fig. 10. Overview of the proposed model.

7.1 Evaluation from a Network Perspective

The properties that are put under investigation provide a representative picture of the corresponding network topology [8], [46], [47], [65] and thus can serve as indicators of how much the predicted network resembles the graph corresponding to the actual software system of the final version. These properties are:

a) the distribution of the distance between all class pairs (captured by the number of class pairs versus the distance in number of hops). For this analysis we employ the aforementioned hop plots.
b) the in-degree distribution of the corresponding graph expressed by the exponent (α) of the power function.
c) the diameter of the corresponding networks (i.e., the longest shortest path between any two nodes, where shortest paths are calculated according to Dijkstra's algorithm [17]).
d) size properties of the corresponding networks (number of nodes, edges and the resulting density) as representative for the accuracy of the proposed model to estimate the growth of the simulated software system.

To provide insight into the benefit of enhancing the simple Preferential Attachment model with the additional parameters and rules shown earlier, the results are given for the last version of the training set (*Base*), for the network derived when applying the Preferential Attachment model, for the network derived when applying all parameters and rules of the proposed model (*Proposed*) and for the actual network corresponding to the last version of the test dataset (*Final*).

The results are shown in Fig. 11 for the distribution of the distance in terms of hop plots and in Table 6 for the numerical values of the power exponent (α), the diameter and the size properties, for all examined software projects. Obviously, the goal of the simulation is to reproduce the final version as accurately as possible.

As it can be observed from the hop plots, in almost all cases the proposed model increases significantly the forecasting power over the simple Preferential Attachment model and bridges the largest part of the distance between the Base and the Final version. From Fig. 11 it becomes clear that the distribution of the distance between class pairs predicted by the proposed model, approaches the distribution for the final version more efficiently than the simple preferential attachment model, for the entire spectrum of distances.

A comparison of the proposed model and the Preferential Attachment model on the basis of the other network properties is shown in Table 6. To enable an evaluation of the relative error, as well as the improvement over the last examined version which served as the baseline for the prediction, the network properties are also shown for the corresponding graphs (Final, Base).

For each case, the graph diameter δ, the number of nodes $|V|$, the number of edges $|E|$, the graph density and the slope α of the degree distribution, are predicted with higher accuracy by the proposed model (the obtained value is closer to that of the final version) than by the simple Preferential Attachment model.

To investigate whether the difference between the proposed and the Preferential Attachment model is statistically significant, we compared their ability to capture the hop plot of the final system. To this end, we compared the errors resulting from each approach by employing a paired samples t-test and a related samples Wilcoxon Test depending on whether the differences of errors are normally distributed or not [45], [56]. The paired samples t-test is ideal for the comparison of a pair of variables distinguished by a Boolean characteristic (e.g., Before, After) [45], [56]. However this test can be applied only if the differences of the pairs are normally distributed. If they are not, the most appropriate test is the Wilcoxon. Consequently, at first we ran a Kolmogorov Smirnov (K-S) test for normality in the differences of the errors and then, depending on the K-S test result we applied either the paired samples t-test or the Wilcoxon test.

The corresponding null hypothesis of the t-test (Wilcoxon test) is that the difference of means (medians) between paired observations is equal to zero, i.e., H_0: $\mu_1 - \mu_2 = 0$. The results are shown in Table 7. As it can be observed for seven out of the ten projects the proposed model is superior to the PA model since the resulting error is lower and the difference is statistically significant. For three projects where the errors are relatively similar, the results are not statistically significant.

A large portion of the improvement over the PA model is due to the consideration of new edges between existing nodes, and new edges between new nodes. Edges of this kind are quite frequent during the evolution of several systems and thus should not be neglected. Moreover, it appears that the consideration of edge removals, in combination with the introduction of edges between existing nodes, helps in modeling accurately the diameter of the predicted network (compared to the base version). It should be mentioned that the percentage of simulated node deletions caused by the removal of edges leading to unconnected nodes is between 0.5 and 1.5 percent.

We have also observed the impact of the application of the domain rules, which in terms of distance distribution helps in following the actual distribution more accurately. In Fig. 12 the impact of considering domain rules on the hop distance among class pairs is shown for project JDeodorant. It becomes evident that when the domain rules are taken into account, the corresponding curve moves closer to that of the final version and moreover the diameter of the final graph is correctly predicted. The simulation revealed that the domain rules regarding the handling of edges emanating from superclasses have been applied many times, which means that otherwise, edges leaving superclass and edges reaching subclasses of the same hierarchy, would have been added throughout the simulated network of classes.

7.2 Evaluation from a Software Perspective

Software prediction models based on the network representation would be valuable if their results could be exploited by software developers or maintainers. As an example, the prediction of future software evolution can indicate which system classes might become overloaded, based on

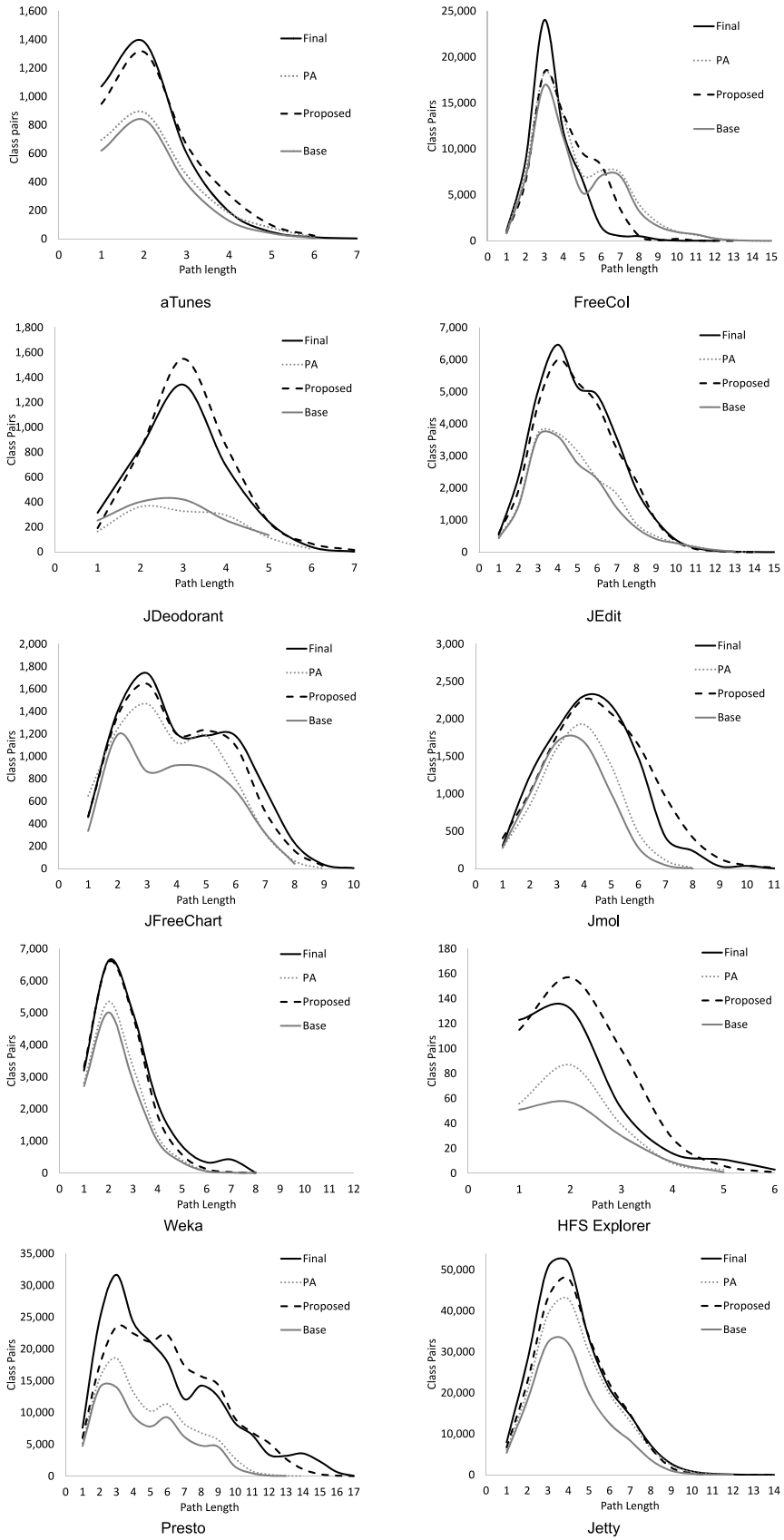


Fig. 11. Comparison of predicted and actual hop plots (curves are smooth for aesthetic reasons: path length can assume only integer values).

their expected in-degree (fan-in). This information might be useful for anticipating possible future "God" or highly-coupled classes.

To investigate how accurately the proposed model can predict the classes with the largest in-degree in a future version, we compare the predictions against the actual

TABLE 6
Comparison of Predicted and Actual Network Properties

		δ	$\|V\|$	$\|E\|$	Density	α
aTunes	Base	6	438	1,004	0.0052	−1.283
	PA	7	473	1,237	0.005541	−1.244
	% error	−22%	−43%	−37%	94%	12%
	Proposed	11	702	1,719	0.003493	−1.276
	% error	22%	−16%	−13%	22%	15%
	Final	9	832	1,979	0.00286	−1.114
FreeCol	Base	15	548	2,362	0.00788	−1.11645
	PA	15	631	3,336	0.008392	−1.17921
	% error	25%	−18%	−16%	26%	15%
	Proposed	13	820	4,313	0.006422	−1.15214
	% error	8%	6%	9%	−4%	13%
	Final	12	772	3,975	0.006678	−1.0218
JDeodorant	Base	5	84	264	0.037866	−1.1589
	PA	6	141	472	0.023911	−1.1329
	% error	−14%	−38%	−39%	60%	−22%
	Proposed	7	238	710	0.012587	−1.3285
	% error	0%	4%	−9%	−16%	−8%
	Final	7	229	779	0.01492	−1.44615
Jedit	Base	13	662	1,499	0.003426	−1.38723
	PA	14	693	1,638	0.003416	−1.38888
	% error	−7%	−28%	−28%	38%	0%
	Proposed	14	1,042	2,290	0.002111	−1.385
	% error	−7%	9%	1%	−14%	0%
	Final	15	960	2,273	0.002469	−1.38898
JFreeChart	Base	8	532	1,733	0.006135	−1.19137
	PA	9	584	2,111	0.0062	−1.13429
	% error	−10%	−43%	−27%	122%	−9%
	Proposed	9	776	2,058	0.003422	−1.18378
	% error	−10%	−24%	−29%	23%	−5%
	Final	10	1,018	2,892	0.002793	−1.24448
Jmol	Base	8	409	845	0.005064	−1.33827
	PA	8	461	971	0.004579	−1.3152
	% error	−33%	−16%	−14%	21%	−8%
	Proposed	13	573	1,184	0.004635	−1.4512
	% error	8%	5%	5%	22%	1%
	Final	12	547	1,130	0.003784	−1.43146
Weka	Base	8	864	3,293	0.004416	−1.03944
	PA	9	1,296	5,198	0.003097	−1.0032
	% error	−18%	−16%	−12%	26%	−5%
	Proposed	12	1,704	6,010	0.002071	−1.0902
	% error	9%	10%	2%	−16%	3%
	Final	11	1,550	5,884	0.002451	−1.05372
HFS Explorer	Base	5	109	213	0.018094	−1.71211
	PA	5	114	232	0.01801	−1.73183
	% error	−29%	−72%	−81%	150%	28%
	Proposed	8	382	1,183	0.006493	−1.28257
	% error	14%	−8%	−4%	−10%	−5%
	Final	7	414	1,233	0.007211	−1.3511
Presto	Base	13	1,380	4,745	0.00249	−1.19893
	PA	14	1,812	5,177	0.00154	−1.23873
	% error	−18%	−18%	−32%	−16%	−2%
	Proposed	17	1,812	6,011	0.001578	−1.31144
	% error	0%	−18%	−21%	−14%	4%
	Final	17	2,219	7,595	0.00183	−1.25875
Jetty	Base	12	2,188	5,396	0.001128	−0.29392
	PA	12	2,902	6,110	0.000726	−1.35826
	% error	−14%	−9%	−23%	−6%	4%
	Proposed	12	2,902	6,754	0.0008	−1.33193
	% error	−14%	−9%	−15%	3%	2%
	Final	14	3,201	7,952	0.000776	−1.30299

evolution. Moreover, we compare the predicted afferent and efferent coupling at the package level to the actual values of the last examined version.

TABLE 7
Statistical Comparison of Errors between the Proposed and the PA Model

Project Name	Error (PA)	Error (Proposed)	Sig. (2-tailed)	Method
aTunes	0.22	0.02	0.031*	t-test
FreeCol	0.18	0.12	0.007*	t-test
JDeodorant	0.005	0.005	0.345	Wilcoxon
JEdit	0.08	0.02	0.05*	t-test
JFreeChart	0.11	0.03	0.012*	Wilcoxon
jMol	0.15	0.07	0.033*	t-test
Weka	0.17	0.02	0.02*	t-test
HFS	0.024	0.016	0.893	Wilcoxon
Presto	0.1385	0.0837	0.007*	t-test
Jetty	0.0337	0.0302	0.216	t-test

implies that the result is statistically significant at the 0.05 level.

In Fig. 13 we illustrate for all projects, the evolution of the in-degree for the class that exhibited the largest variation between the first and last examined version (the same classes as in Fig. 2). Each figure shows the actual evolution of the in-degree (continuous line, where the thicker part corresponds to the training period), the evolution predicted by the proposed model (dashed line), the PA model (grey line) and the regression-based model (dotted line). The three models employed for training all versions up to the version denoted as Base.

As it can be observed, despite the numerous factors that affect the growth of a software system, the variation of the in-degree is captured to a large extent by the proposed model offering significant improvement over the PA and the regression models. The improved performance of the proposed graph-based prediction model, justifies the use of graphs as means for the study of evolution, since it becomes evident that simple regression is inadequate. Moreover, the improved results compared to the ones of the preferential attachment model are due to the consideration of the various parameters in our model.

For the shown classes the average error in the prediction of the actual in-degree is 7.3 percent for the proposed model, while for the PA and regression models the error is 19.1 and 24 percent, respectively. Certainly, a prediction model cannot guarantee to reveal all forthcoming changes in software but providing even a relatively

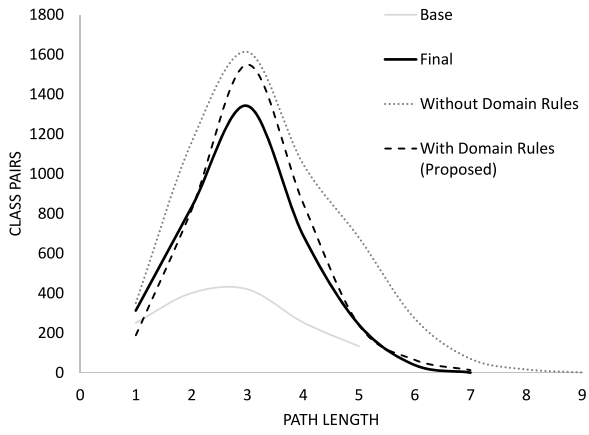


Fig. 12. Impact of domain knowledge application.

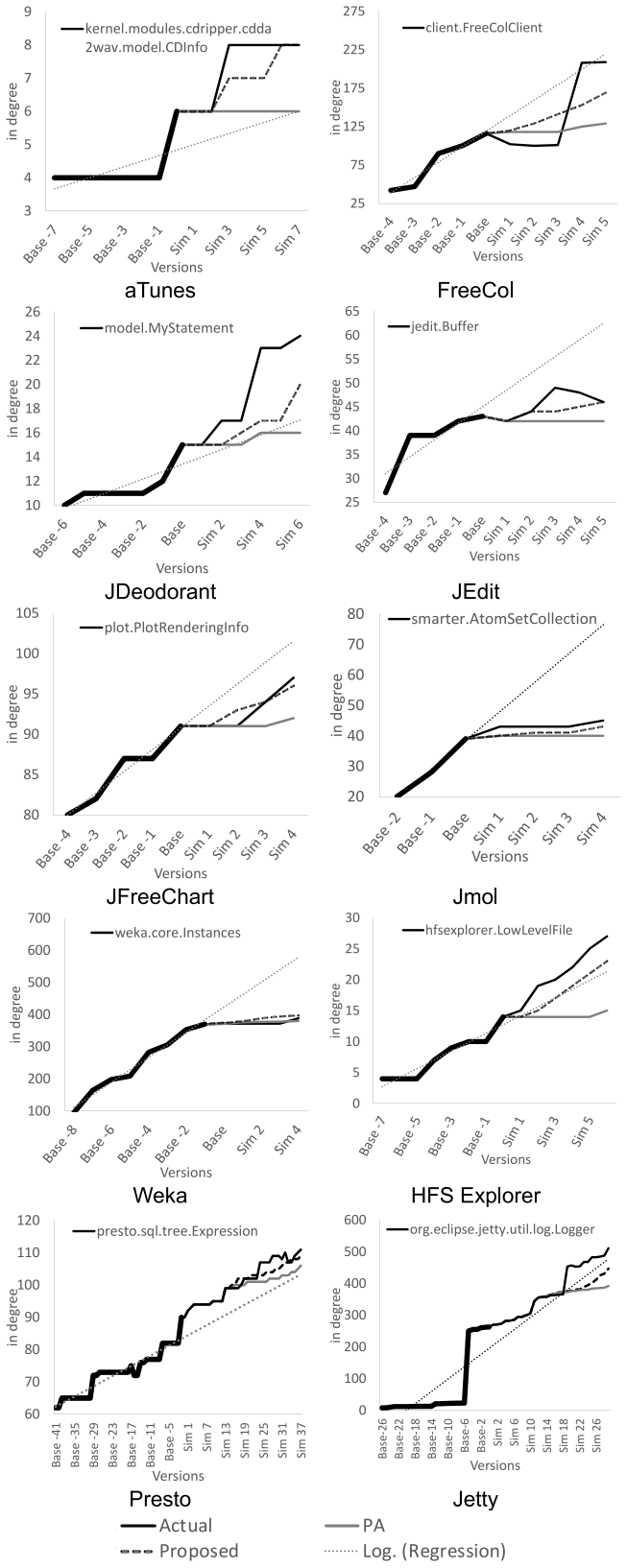


Fig. 13. Comparison of actual and predicted evolution of in-degree.

accurate prediction might be the basis for further work in this direction.

As already mentioned, in contrast to all existing models, the proposed prediction model considers also edge removals. Such edge removals, although less frequent than edge additions, might result in decreases of the in- and out-degree of particular nodes. We have measured the percentage of classes whose in- and out-degree dropped between the first and last examined version. The percentages vary from 0 to 2.7 percent for in-degree reductions and from 0.6 to 12 percent for out-degree reductions. To assess the ability of the proposed model in predicting cases where the degree of a node decreases, Table 8 shows the number of classes whose in- and out-degree actually decreased during the evolution of the examined systems as well as the classes for which the proposed model successfully predicted a degree reduction.

TABLE 8
Prediction of Classes Whose In-and Out-degree Decreased

Project Name	Classes whose in-degree decreased		Classes whose out-degree decreased	
	Actual	Predicted	Actual	Predicted
aTunes	13	5	31	28
FreeCol	21	6	11	3
JDeodorant	5	5	5	2
JEdit	17	9	26	16
JFreeChart	8	7	36	36
jMol	8	1	9	1
Weka	1	1	1	1
HFS	0	0	0	0
Presto	52	37	33	28
Jetty	41	29	19	16

* *Matching has been performed at the level of identical class names.*

A network-based software prediction model can also be useful in anticipating the evolution of more 'traditional' software properties, such as the afferent and efferent coupling at the package level. In our context, the afferent coupling for a given package refers to the number of other packages that depend upon classes within this package and can be considered as an indicator of the package's responsibility. Efferent coupling refers to the number of other packages that the classes in a given package depend upon and can be considered as an indicator of the package's independence [54]. In order to evaluate the accuracy of the proposed model we investigated the predicted afferent and efferent package coupling. Fig. 14 depicts graphically the afferent and efferent coupling for a) the last version of the training set (Base), b) the last version of the test dataset (Final) and c) the results predicted by the proposed model (Proposed), for all examined systems. The set of bars on the left-hand side correspond to the afferent coupling and values are measured against the bottom x-axis. The set of bars on the right-hand side correspond to the efferent coupling and values are measured against the top x-axis. The packages correspond to the top 10 packages with the highest afferent coupling in the Final version (in case the system contains less than 10 packages, all are shown). In cases where only one bar is shown (representing the Final version), the corresponding package had a zero coupling value in the base version.

In general, it can be observed that in a large number of cases where the coupling value of the final version is larger than that of the base version, the proposed model correctly

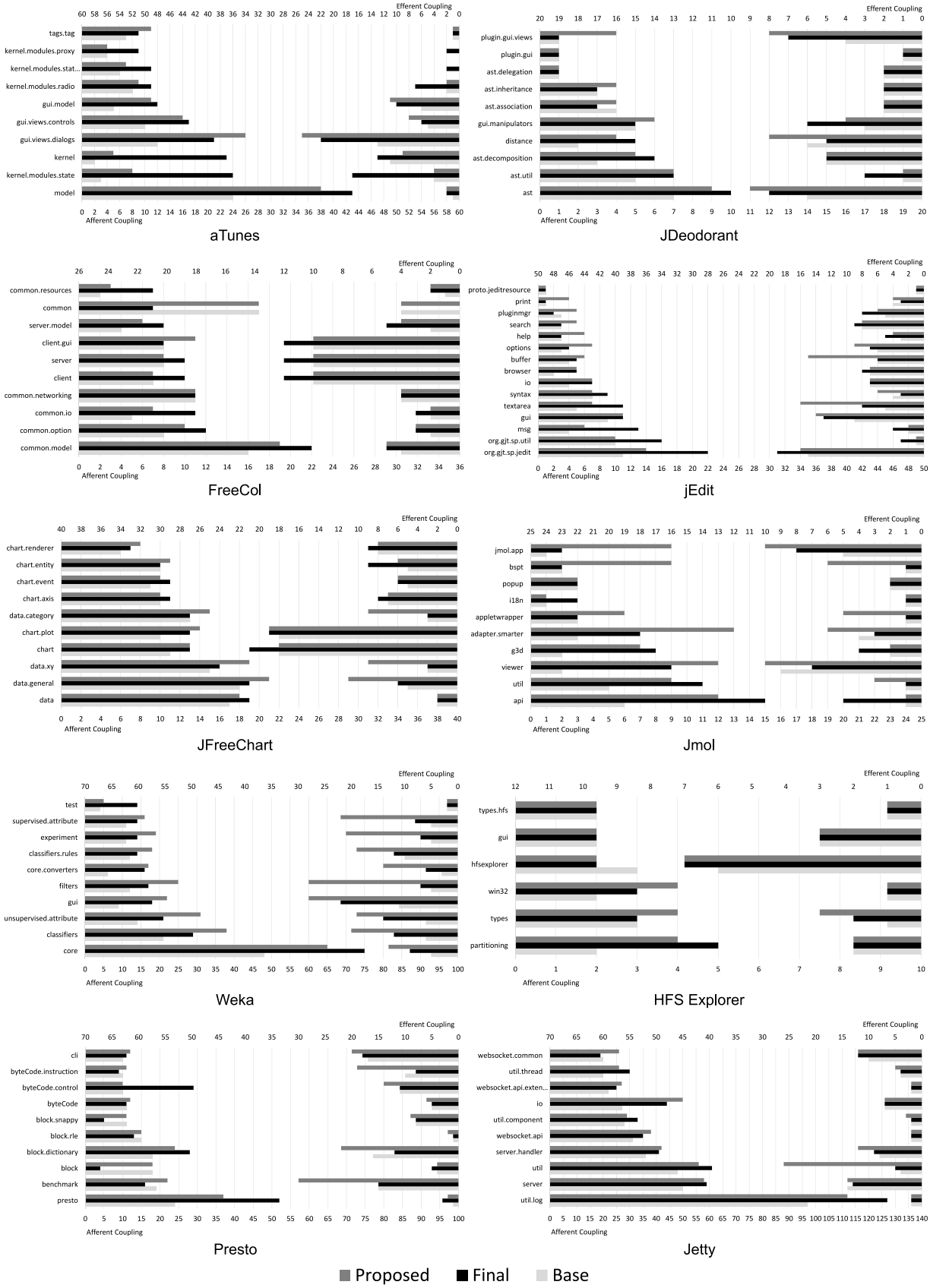


Fig. 14. Comparison of predicted Afferent and Efferent Package Couplings. Afferent Coupling is illustrated on the left bars (bottom *x*-axis). Efferent Coupling is illustrated on the right bars (top *x*-axis).

identified this increasing trend and predicted a value that is larger than that of the initial state. In particular, changes in the afferent coupling have been correctly predicted in 81 percent of the cases, while changes in the efferent coupling have been correctly predicted in 60 percent of the cases.

However, as it can be observed, there are several cases where the accuracy of the prediction is limited. This is primarily the case when the set of historical observations is not sufficient. As an example, consider the case of package modules.radio in project aTunes (fourth set of bars on the right of the first figure). The predicted efferent coupling of value 2 is equal to the efferent coupling that the project had at the base version (end of training period) while the actual coupling at the end of the forecasting period has risen up to 7. The reason is that this package was introduced into the system in the last version of the training period thus offering no historical data to guide the prediction. As already mentioned, the quality of a forecast depends heavily on the representativeness of the recorded history.

The aforementioned results indicate that a network prediction model with sufficient forecasting power can be valuable in guiding the maintenance efforts of a software project. Anticipating which classes exhibit a tendency to become large in terms of functionality offered to clients or extremely dependent upon other classes, might be helpful to identify system modules that warrant further attention and possibly refactoring. Similarly, it would be valuable to know in advance which packages might develop excessive coupling among their modules or to other packages and which parts of the architecture appear to be volatile. In any case, representing software systems in terms of networks enables a systematic modeling of their evolution and can provide a glimpse into their future. Furthermore, the output of this kind of analysis can be fed to empirical studies that would investigate the relation among the examined network properties and design qualities of the underlying systems in order to validate the use of graph metrics as indicators of software quality.

8 Limitations and Threats to Validity

The accuracy of any prediction model is by definition constrained, especially in the software domain, since future evolution can be affected by numerous, unpredictable factors such as requirements for implementing radically novel functionality, decisions to modify the architecture, changes in the development team etc. The limitations and threats to validity that can be identified are outlined next.

The proposed approach samples from distributions based on past version data to obtain several model parameters. This sampling assumes that the system under study will continue to evolve in the future in a similar manner as in the past. This means that if abrupt changes to the network topology occur due to major architectural modifications for example, the predicted future network evolution might be inaccurate. As another example, if the trend in the number of nodes and edges has been increasing during the entire history of a software project with an exception for the last version (where it could possibly drop), the model would not be able to predict this sudden reduction in the number of nodes and edges. Another source of inaccuracy is related to the fact that in order to derive the model parameters, the distributions are captured by means of curve fitting to the actual data. The corresponding mathematical function might not always have an ideal fit to the actual data points, affecting the accuracy of the extracted parameter.

Apart from the aforementioned limitations, the proposed prediction model suffers from the usual threats to external and internal validity. The fact that the model is evaluated against 10 projects, unavoidably limits the possibility to extensively generalize our findings. This threat is related to the observation of general trends which are applicable to all projects (such as preferential attachment). It is always possible that another set of projects might exhibit different phenomena. A similar threat stems from the fact that all analyzed projects are developed in Java thus limiting the ability to generalize to other object-oriented languages. However, since a large part of the model consists in learning from past versions, we believe that it can be successfully adapted to any software project regardless of its particularities.

Concerning the internal validity (i.e., the parameters that might affect the evolutionary trends that we are trying to predict), it is reasonable to assume that numerous other factors, which affect software growth, might have not been taken into consideration. As an example, a forecasting model could be augmented by considering the co-evolving developer community or dependencies among documented requirements. However, the proposed simulation approach can be considered as incremental in the sense that additional parameters can be easily integrated. The same holds for the incorporation of additional domain rules which might be representative for a particular application domain or the habits of a particular development team.

9 Conclusions and Future Work

By drawing analogies between networks in other domains and software, we have attempted to develop a prediction model that is capable of forecasting trends in the evolution of networks representing Java systems. The model incorporates findings regarding the growth patterns of software networks, such as the conformance to the preferential attachment and the duplication model. Moreover, crucial model parameters are extracted by sampling from distributions formed by analyzing past versions and therefore the prior evolution is taken into account. Finally, acknowledging the importance of considering domain knowledge, the proposed model has been enhanced by rules specific to object-oriented design. Evaluation against 10 open-source projects showed that the forecasting can provide sufficient insight to the future evolution trends of software. Software maintenance can benefit from the application of such models by focusing on parts of the network whose properties tend to deteriorate.

Future work includes the automated analysis of projects directly from the repositories in which they reside. This will enable the analysis of an even larger set of projects and versions. Moreover, the incorporation of additional parameters, such as the investigation of developer collaboration networks, will improve the expressiveness of the model and enable the parallel study of the co-evolution between

software artifacts and networks formed by people. Finally, given that software systems often undergo major changes, another line of future research would be to monitor disruption during software evolution to improve prediction in these cases.

ACKNOWLEDGMENTS

This research has been cofinanced by the European Union (European Social Fund—ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF)—Research Funding Program: Thalis—Athens University of Economics and Business—SOFTWARE ENGINEERING RESEARCH PLATFORM. Theodore Chaikalis is the corresponding author.

REFERENCES

[1] Forecasting trends in software evolution: Supplemental material [Online]. Available: http://tinyurl.com/softevolmodel, Oct. 2014.
[2] G. Antoniol, G. Casazza, M. Di Penta, and E. Merlo, "Modeling clones evolution through time series," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2001, pp. 273–280.
[3] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a Java legacy system," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng.*, New York, NY, USA, 2006, pp. 8–17.
[4] aTunes, (2013, Oct.). aTunes audio player and organizer [Online]. Available: http://www.atunes.org
[5] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.
[6] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Softw. Eng.*, vol. 22, no. 10, pp. 751–761, Oct. 1996.
[7] J. Bevan, E. J. Whitehead Jr., S. Kim, and M. Godfrey, "Facilitating software evolution research with Kenyon," in *Proc. 10th Eur. Softw. Eng. Conf. 13th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, New York, NY, USA, 2005, pp. 177–186.
[8] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *Proc. 34th Int. Conf. Softw. Eng.*, 2012, pp. 419–429.
[9] C. K. Chang, H. Jiang, Y. Di, D. Zhu, and Y. Ge, "Time-line based model for software project scheduling with genetic algorithms," *Inf. Softw. Technol.*, vol. 50, no. 11, pp. 1142–1154, Oct. 2008.
[10] F. Chung, L. Lu, T. G. Dewey, and D. J. Galas, "Duplication models for biological networks," *J. Comput. Biol. J. Comput. Mol. Cell Biol.*, vol. 10, no. 5, pp. 677–687, 2003.
[11] F. R. K. Chung and L. Lu, *Complex Graphs and Networks*. Providence, RI, USA: Amer. Math. Soc., 2006.
[12] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, pp. 44–49, Aug. 1994.
[13] G. Concas, M. Marchesi, S. Pinna, and N. Serra, "Power-laws in a large object-oriented software system," *IEEE Trans. Softw. Eng.*, vol. 33, no. 10, pp. 687–708, Oct. 2007.
[14] M. D'Ambros and M. Lanza, "A flexible framework to support collaborative software evolution analysis," in *Proc. 12th Eur. Conf. Softw. Maintenance Reeng.*, 2008, pp. 3–12.
[15] J. K. Das, *Statistics for Business Decisions*. San Francisco, CA, USA: Academic.
[16] L. Devroye, *Non-Uniform Random Variate Generation*. New York, NY, USA: Springer-Verlag, 1986.
[17] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.
[18] D. Easley and J. Kleinberg, *Networks, Crowds, and Markets Reasoning about a Highly Connected World*. New York, NY, USA: Cambridge Univ. Press, 2010.
[19] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, New York, NY, USA, 1999, pp. 251–262.
[20] M. A. Fortuna, J. A. Bonachela, and S. A. Levin, "Evolution of a modular software network," *Proc. Nat. Acad. Sci. USA*, vol. 108, pp. 19985–19989, Nov. 2011.
[21] FreeCol. (2013, Oct.). *FreeCol—The Colonization of America* [Online]. Available: http://www.freecol.org/
[22] M. Genero, J. Olivas, M. Piattini, and F. Romero, "Using metrics to predict OO information systems maintainability," in *Advanced Information Systems Engineering*, K. R. Dittrich, A. Geppert, and M. C. Norrie, Eds. Berlin, Germany: Springer, 2001, pp. 388–401.
[23] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*. Upper Saddle River, NJ, USA: Prentice-Hall, 2003.
[24] T. Gîrba and S. Ducasse, "Modeling history to analyze software evolution: Research articles," *J. Softw. Maintenance Evol.*, vol. 18, no. 3, pp. 207–236, May 2006.
[25] T. Gîrba, S. Ducasse, and M. Lanza, "Yesterdays weather: Guiding early reverse engineering efforts by summarizing the evolution of changes," in *Proc. Int. Conf. Softw. Maintenance*, 2004, pp. 40–49.
[26] M. W. Godfrey and D. M. German, "The past, present, and future of software evolution," in *Proc. Frontiers Softw. Maintenance*, 2008, pp. 129–138.
[27] HFS Explorer. (2013, Oct.) *HFS Explorer* [Online]. Available: http://www.catacombae.org/hfsx.html
[28] S. M. H. Hosseini and S. R. Kesler, "Influence of choice of null network on small-world parameters of structural correlation networks," *PLoS ONE*, vol. 8, no. 6, p. e67354, Jun. 2013.
[29] JDeodorant. (2014, Dec.). JDeodorant—When quality matters the most! [Online]. Available: http://www.jdeodorant.com/
[30] JEdit. (2013, Oct.). jEdit—Programmer's Text Editor [Online]. Available: http://www.jedit.org/
[31] S. Jenkins and S. R. Kirk, "Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution," *Inf. Sci.*, vol. 177, no. 12, pp. 2587–2601, Jun. 2007.
[32] Jetty. (2014, Nov.). Jetty—Servlet Engine and HTTP Server [Online]. Available: http://www.eclipse.org/jetty/
[33] JFreeChart. (2013, Oct.). JFreeChart [Online]. Available: http://www.jfree.org/jfreechart
[34] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," in *Proc. IEEE/ACM 28th Int. Conf. Autom. Softw. Eng.*, 2013, pp. 279–289.
[35] Jmol. (2013, Oct.). *Jmol: An open-source Java viewer for chemical structures in 3D* [Online]. Available: http://www.jmol.org/
[36] M. Jørgensen, "Forecasting of software development work effort: Evidence on expert judgement and formal models," *Int. J. Forecasting*, vol. 23, no. 3, pp. 449–462, Jul. 2007.
[37] E. S. Jun and J. K. Lee, "Quasi-optimal case-selective neural network model for software effort estimation," *Expert Syst. Appl.*, vol. 21, no. 1, pp. 1–14, Jul. 2001.
[38] H. Kagdi, "Improving change prediction with fine-grained source code mining," in *Proc. 22nd IEEE/ACM Int. Conf. Autom. Softw. Eng.*, New York, NY, USA, 2007, pp. 559–562.
[39] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting faults from cached history," in *Proc. 29th Int. Conf. Softw. Eng.*, Washington, DC, USA, 2007, pp. 489–498.
[40] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proc. 32nd ACM Symp. Theory Comput.*, 2000, pp. 163–170.
[41] K. A. Kontogiannis, R. Demori, E. Merlo, M. Galler, and M. Bernstein, "Pattern matching for clone and concept detection," *Autom. Softw. Eng.*, vol. 3, no. 1/2, pp. 77–108, Jun. 1996.
[42] C. van Koten and A. R. Gray, "An application of Bayesian network for predicting object-oriented software maintainability," *Inf. Softw. Technol.*, vol. 48, no. 1, pp. 59–67, Jan. 2006.
[43] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, "Stochastic models for the web graph," in *Proc. 41st Annu. Symp. Found. Comput. Sci*, 2000, pp. 57–65.
[44] M. M. Lehman, "Programs, life cycles, and laws of software evolution," *Proc. IEEE*, vol. 68, no. 9, pp. 1060–1076, Sep. 1980.
[45] E. L. Lehmann and J. P. Romano, *Testing Statistical Hypotheses*. New York, NY, USA: Springer, 2005.
[46] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins, "Microscopic evolution of social networks," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, New York, NY, USA, 2008, pp. 462–470.
[47] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. ACM SIGKDD Knowl. Discovery Data Mining*, 2005, pp. 177–187.

[48] K. J. Lieberherr and I. Holland, "Assuring good style for object-oriented programs," *IEEE Softw.*, vol. 6, no. 5, pp. 38–48, Sep. 1989.
[49] H. Li, H. Zhao, W. Cai, J.-Q. Xu, and J. Ai, "A modular attachment mechanism for software network evolution," *Phys. Stat. Mech. Appl.*, vol. 392, no. 9, pp. 2025–2037, May 2013.
[50] J. Li, G. Ruhe, A. Al-Emran, and M. M. Richter, "A flexible method for software effort estimation by analogy," *Empirical Softw. Eng.*, vol. 12, no. 1, pp. 65–106, Feb. 2007.
[51] L. Li, D. Alderson, J. C. Doyle, and W. Willinger, "Towards a theory of scale-free graphs: Definition, properties, and implications," *Internet Math.*, vol. 2, p. 4, 2005.
[52] Y. F. Li, M. Xie, and T. N. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," *J. Syst. Softw.*, vol. 82, no. 2, pp. 241–252, Feb. 2009.
[53] P. Louridas, D. Spinellis, and V. Vlachos, "Power laws in software," *ACM Trans. Softw. Eng. Methodol.*, vol. 18, no. 1, pp. 2:1–2:26, Oct. 2008.
[54] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ, USA: Prentice-Hall, 2003.
[55] Y. Ma, K. He, and J. Liu, "Network motifs in object-oriented software systems," Dynamics of Continuous, Discrete and Impulsive Systems, vol. 14, pp. 166–172, 2007.
[56] J. McDonald, *Handbook of Biological Statistics*. Baltimore, MD, USA: Sparky House Publishing, 2009.
[57] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, New York, NY, USA, 2008, pp. 13–23.
[58] T. Mens, "Introduction and roadmap: History and challenges of software evolution," in *Software Evolution*, Berlin, Germany: Springer, 2008, pp. 1–11.
[59] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs," *Phys. Rev. E*, vol. 68, no. 4, p. 046116, Oct. 2003.
[60] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proc. 27th Int. Conf. Softw. Eng.*, New York, NY, USA, 2005, pp. 284–292.
[61] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proc. 28th Int. Conf. Softw. Eng.*, New York, NY, USA, 2006, pp. 452–461.
[62] N. K. Nagwani and A. Bhansali, "A data mining model to predict software bug complexity using bug estimation and clustering," in *Proc. Int. Conf. Recent Trends Inf., Telecommun. Comput.*, 2010, pp. 13–17.
[63] P. Paymal, R. Patil, S. Bhowmick, and H. Siy, "Measuring disruption from software evolution activities using graph-based metrics," in *Proc. 27th IEEE Int. Conf. Softw. Maintenance*, 2011, pp. 532–535.
[64] P. C. Pendharkar, G. H. Subramanian, and J. Rodger, "A probabilistic model for predicting software development effort," *IEEE Trans. Softw. Eng.*, vol. 31, no. 7, pp. 615–624, Jul. 2005.
[65] J. J. Pfeiffer III, T. La Fond, S. Moreno, and J. Neville, "Fast generation of large scale social networks with clustering," *ArXiv12024805 Phys.*, Feb. 2012, http://arxiv.org/abs/1202.4805
[66] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Proc. 16th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, New York, NY, USA, 2008, pp. 2–12.
[67] A. Potanin, J. Noble, M. Frean, and R. Biddle, "Scale-free geometry in OO programs," *Commun. ACM*, vol. 48, no. 5, pp. 99–103, May 2005.
[68] Presto. (2014, Nov.). *Presto | Distributed SQL Query Engine for Big Data* [Online]. Available: http://prestodb.io/
[69] U. Raja and M. J. Tretter, "Defining and evaluating a measure of open source project survivability," *IEEE Trans. Softw. Eng.*, vol. 38, no. 1, pp. 163–174, Jan./Feb. 2012.
[70] A. J. Riel, *Object-Oriented Design Heuristics*. Reading, MA, USA: Addison-Wesley, 1996.
[71] K. Shibata, K. Rinsaka, T. Dohi, and H. Okamura, "Quantifying software maintainability based on a fault-detection/correction model," in *Proc. 13th Pacific Rim Int. Symp. Dependable Comput.*, 2007, pp. 35–42.
[72] S. Sun, C. Xia, Z. Chen, J. Sun, and L. Wang, "On structural properties of large-scale software systems: From the perspective of complex networks," in *Proc. 6th Int. Conf. Fuzzy Syst. Knowl. Discovery*, 2009, vol. 7, pp. 309–313.
[73] C. Taube-Schock, R. J. Walker and I. H. Witten, "Can we avoid high coupling?" in *Proc. 25th Eur. Conf. Object-Oriented Programm.*, 2011, pp. 204–228.
[74] I. Turnu, G. Concas, M. Marchesi, S. Pinna, and R. Tonelli, "A modified Yule process to model the evolution of some object-oriented system properties," *Inf. Sci.*, vol. 181, no. 4, pp. 883–902, Feb. 2011.
[75] I. Turnu, G. Concas, M. Marchesi, and R. Tonelli, "The fractal dimension of software networks as a global quality metric," *Inf. Sci.*, vol. 245, pp. 290–303, Oct. 2013.
[76] *COCOMO II Model Definition Manual*, USC-CSE, Center Softw. Eng., Comput. Sci. Dept., Southern California, Los Angeles, CA, USA, 1997.
[77] S. Valverde and R. V. Sole, "Hierarchical small worlds in software architecture," *ArXivcond-Mat0307278*, Jul. 2003, http://arxiv.org/abs/cond-mat/0307278
[78] R. Vasa, J.-G. Schneider, and O. Nierstrasz, "The inevitable stability of software change," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2007, pp. 4–13.
[79] K. Vinay Kumar, V. Ravi, M. Carr, and N. Raj Kiran, "Software development cost estimation using wavelet neural networks," *J. Syst. Softw.*, vol. 81, no. 11, pp. 1853–1867, Nov. 2008.
[80] L. Wang, Z. Wang, C. Yang, L. Zhang, and Q. Ye, "Linux kernels as complex networks: A novel method to study evolution," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2009, pp. 41–50.
[81] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun. 1998.
[82] Weka. (2013, Oct.). Weka: Data mining software in Java [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/
[83] R. Wettel, "Visual exploration of large-scale evolving software," in *Proc. 31st Int. Conf. Softw. Eng. Companion Volume*, 2009, pp. 391–394.
[84] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proc. 33rd Int. Conf. Softw. Eng.*, New York, NY, USA, 2011, pp. 551–560.
[85] R. Wheeldon and S. Counsell, "Power law distributions in class relationships," in *Proc. 3rd IEEE Int. Workshop Source Code Anal. Manipulation*, 2003, pp. 45–54.
[86] F. G. Wilkie and B. A. Kitchenham, "Coupling measures and change ripples in C++ application software," *J. Syst. Softw.*, vol. 52, nos. 2/3, pp. 157–164, Jun. 2000.
[87] W. Willinger, D. Alderson, and J. C. Doyle, "Mathematics and the internet: A source of enormous confusion and great potential," *Notices Amer. Math. Soc.*, vol. 56, no. 5, pp. 586–599, 2009.
[88] H. S. Yazdi, M. Mirbolouki, P. Pietsch, T. Kehrer, and U. Kelter, "Analysis and prediction of design model evolution using time series," in *Proc. Adv. Inf. Syst. Eng. Workshops*, 2014, pp. 1–15.
[89] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Trans. Softw. Eng.*, vol. 30, no. 9, pp. 574–586, Sep. 2004.
[90] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using OO metrics. An industrial case study," in *Proc. 6th Eur. Conf. Softw. Maintenance Reeng.*, 2002, pp. 99–107.
[91] M. S. Zanetti and F. Schweitzer, "A network perspective on software modularity," in ARCS Workshops (ARCS), Muenchen, pp. 1–8, Jan. 2012.
[92] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: An empirical study of commercial software projects," in *Proc. Int. Conf. Softw. Eng.*, Piscataway, NJ, USA, 2013, pp. 1042–1051.
[93] X. Zheng, D. Zeng, H. Li, and F. Wang, "Analyzing open-source software systems as complex networks," *Phys. Stat. Mech. Appl.*, vol. 387, no. 24, pp. 6190–6200, Oct. 2008.
[94] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," *J. Syst. Softw.*, vol. 80, no. 8, pp. 1349–1361, Aug. 2007.
[95] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proc. 30th Int. Conf. Softw. Eng.*, New York, NY, USA, 2008, pp. 531–540.
[96] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proc. 3rd Int. Workshop Predictor Models Softw. Eng*, Washington, DC, USA, 2007, p. 9.

Theodore Chaikalis received the BSc and MSc degrees in applied informatics from the University of Macedonia, in 2007 and 2009, respectively. He is currently working toward the PhD degree in the Department of Applied Informatics at the University of Macedonia under the supervision of Dr. Alexander Chatzigeorgiou. His research interests include object-oriented design, object-oriented quality metrics, graph theory and software evolution analysis and simulation. He is a member of the IEEE and the IEEE Computer Society.

Alexander Chatzigeorgiou received the diploma in electrical engineering and the PhD degree in computer science from the Aristotle University of Thessaloniki, Greece, in 1996 and 2000, respectively. He is an associate professor of software engineering in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. From 1997 to 1999, he was with Intracom S.A., Greece, as a telecommunications software designer. Since 2007, he is also a member of the teaching staff at the Hellenic Open University. His research interests include object-oriented design, software maintenance, and software evolution analysis. He is a member of the IEEE and the Technical Chamber of Greece.