Predicting Software Defectiveness through Network Analysis

*Matteo Orrú
matteo.orru@diee.unica.it

+Cristina Monni
cristina.monni@usi.ch

*Michele Marchesi
michele@diee.unica.it

*Giulio Concas
concas@diee.unica.it

*Roberto Tonelli
roberto.tonelli@dsf.unica.it

*Department of Electrical and Electronic Engineering (DIEE)
University of Cagliari
Piazza D'Armi, 09123 Cagliari (Italy)

+ Faculty of Informatics
University of Lugano
Via Giuseppe Buffi, 13
Lugano, Switzerland 6900

Abstract

We used a complex network approach to study the evolution of a large software system, Eclipse, with the aim of statistically characterizing software defectiveness along the time. We studied the software networks associated to several releases of the system, focusing our attention specifically on their community structure, modularity and clustering coefficient. We found that the maximum average defect density is related, directly or indirectly, to two different metrics: the number of detected communities inside a software network and the clustering coefficient. These two relationships both follow a power-law distribution which leads to a linear correlation between clustering coefficient and number of communities. These results can be useful to make predictions about the evolution of software systems, especially with respect to their defectiveness.

Copyright © 2015 by the paper's authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors.

In: A.H. Bagge, T. Mens (eds.): Postproceedings of SATToSE 2015 Seminar on Advanced Techniques and Tools for Software Evolution, University of Mons, Belgium, 6-8 July 2015, published at `http://ceur-ws.org`

1 Introduction

Modern software systems are large and complex products, built according to a modular structure, where modules (like classes in object oriented systems) are connected with each other to enable software reuse, encapsulation, information hiding, maintainability and so on. Software modularization is acknowledged as a good programming practice [Par72, BC99, SM96] and a certain emphasis is put on the prescription that design software with low coupling and high cohesion would increase its quality [CK94]. In this work we present a study on the relationships between the quality of software systems and their modular structure. To perform this study we used an approach based on the concept of complex networks.

Due to the fact that software systems are inherently complex, the best model to represent them is by retrieving their associated networks [Mye03, ŠB11, WKD07, ŠŽBB15, ZN08]. In other words, in a software network, nodes can be associated to software modules (e.g. classes) and edges can be associated to connections between software modules (e.g. inheritance, collaboration relationships). We investigated the software modular structure - and its impact on software defectiveness - by studying specific network properties: community structure, modularity and clustering coefficient.

A community inside a network is a subnetwork formed by nodes that are densely connected if compared to nodes outside the community [GN01]. Mod-

ularity is a function that measures how marked is a community structure, namely the way the nodes are arranged in communities [NG04]. The clustering coefficient is a measure of connectedness among the nodes of a network [New03].

We studied several releases of a large software system, Eclipse, performing a longitudinal analysis of the relationship between community structure, clustering coefficient and software defectiveness. Our aim is to figure out if the studied metrics can be used to better understand the evolution of software defectiveness along the time and to predict the defectiveness of future releases. The results shown in this paper are part of a more extensive research on different Java projects, which is currently under consolidation. The aim of the authors is to show eventually that the results presented in this work are valid also for other extensively used Java projects.

This paper is organized as follows. In Section 2 we review some recent literature on software network analysis, community structure and defect prediction. In Section 3 we introduce some background concepts taken from the research on complex networks, whereas in Section 4 we thoroughly report the adopted metrics and the methodology. In Section 5 we present some of our results and discuss them in Section 6. In Section 7 we illustrate the threats to validity and in Section 8 we draw some conclusions and outline the future work.

2 Related Work

Being software systems often large and complex, one of the best candidates to represent them is the complex network [FMS01], [Mye03], [TCM$^+$11]. Many software networks, like class diagrams [RS02, VS03], collaboration graph [Mye03], package dependencies networks [CL04], have already been shown to present the typical properties of complex networks [BAJ00], such as fractal and self-similar features [TMT12], scale free [CS05], and small world properties, and consequently power law distributions for the node degree, for the bugs [CMM$^+$11] and for refactored classes [MTM$^+$12].

Modelling a software system as a complex network has been shown to have many applications to the study of failures and defects. For example, Wen [WKD07] reported that scale-free software networks could be more robust if compared to random failures. Other methods have been applied to understand the relationship between number of defects and LOC [Zha09], while in Ostrand et al. a negative binomial regression model is used to show that the majority of bugs is contained only in a fraction of the files (20%) [OWB05].

So far many other methods have been tried for bug prediction [DLR10, HBB$^+$12], especially using dependency graphs [NAH10, ZN08], but only recently many researchers focused their attention on the community structure as defined in social network analysis, namely the division in subgroups of nodes among which there is a high density of connections if compared to nodes that are outside the community [NG04]. Being more connected, elements belonging to the same community might represent functional units or software modules, leading to practical applications of the community detection in the software engineering field. Community detection is usually performed with methods like hierarchical clustering and partitional clustering [For10].

Newman et al. proposed some algorithms for community detection [NG04, New04, CNM04], which are now extensively used in the literature, along with the definition *modularity*, a quality function which measures the strength of a network partition in communities [NG04]. In this work we use one of the algorithms proposed by Newman et al. to understand if such division can be related to software modularity as defined in software engineering and, eventually, if the community metrics may be useful to predict bugs in future releases. The issue of community structure and its application to software engineering has been recently addressed in a similar fashion by Šubelj and Bajek. The authors applied some community detection algorithms to several Java systems to show that their evident community structure does not correspond to the package structure devised by the designer [ŠB11].

In this work we consider the concept of modularity used in software engineering, that is often associated to high values of cohesion and low values of coupling metrics [MMCG99, MM07, ASTC11]. In the literature there are previous attempts to use software network theory to characterize a modularity function and relate it to good programming practice [MMCG99, MM07, ASTC11]. We show that a modularity function based on pure network topology can be used to assess the goodness of a division in clusters, and it is related to the software engineering concept of the separation of components. Our work uses a methodology based on community structure, which is very lightweight from the computational point of view. We are also introducing for the first time some concepts from social network analysis that allowed us to draw the same conclusions of the authors of the aforementioned papers, but getting also information on the predictability of software defectiveness.

3 Modularity and Community Structure

The concept of community derives from social networks. Nodes belonging to the same community are densely connected among each other, while they are poorly connected with nodes which are not in the same

community. Inside a network, a community structure is the specific way in which the nodes are arranged in communities [For10]. Since there can be more than one community structure, we need a quantitative measure to evaluate the best division. The first and most used measure is the modularity [NG04]. Although there are some *caveats* to take into account while using it [GMC10, FB07], modularity is considered the standard measure for the quality of a community structure.

The original definition is based on the fact that a random graph does not possess a community structure, hence providing a null model for comparison with the community structure of real networks [NG04]. Consider a complex network of n nodes and m edges. In order to represent it we can use the following definitions:

- Adjacency matrix:

$$A_{vw} = \begin{cases} 1 & \text{if } v \text{ and } w \text{ are joined} \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

  where v and w are two nodes belonging to communities c_v and c_w;

- Number of edges:

$$m = \frac{1}{2} \sum_{vw} A_{vw}, \tag{2}$$

- Node degree, namely the number of its connected edges:

$$k_v = \sum_{w} A_{vw}. \tag{3}$$

If we postulate that nodes are grouped in communities, then we can compute the fraction among within-community edges and across-communities edges. In order to have a significant community structure this fraction has to be large. Given two communities c_v and c_w, the latter fraction can be written as follows:

$$\frac{\sum_{vw} A_{vw}\, \delta(c_v, c_w)}{\sum_{vw} A_{vw}} = \frac{1}{2m} \sum_{vw} A_{vw}\, \delta(c_v, c_w), \tag{4}$$

where $\delta(c_v, c_w)$ is the Kronecker δ. In order to obtain a reliable measure we need to compare the previous values to a null model. The most used null model is a graph with the same community structure but random connections among its nodes. The expected value of the fraction of edges attached to nodes in community v and to nodes in community w, in the random case would be given by:

$$\frac{1}{2m} \sum_{vw} \frac{k_v k_w}{2m} \delta(c_v, c_w). \tag{5}$$

Subtracting (5) to (4), we get the modularity as defined in Newman [New06]:

$$Q = \frac{1}{2m} \sum_{vw} \Big(A_{vw} - \frac{k_v k_w}{2m} \Big) \delta(c_v, c_w). \tag{6}$$

A good community structure corresponds to values of Q as close as possible to 1. However, in real networks, modularity values that reveal a good community structure fall in a range from 0.3 to 0.7 [NG04]. Lower values are associated to a weak community structure, whereas strong community structures, although rare in practice, may have modularity values higher than 0.7 and approaching 1.

4 Experimental Setting

In this work we analyze the structure and evolution of Eclipse IDE, a popular software system written in Java, using its associated software network. We first retrieved the network associated to each software system - specifically to each subproject in which the major system is structured, by parsing their source code retrieved from their corresponding Source Control Manager (SCM), looking for relationships like collaboration, inheritance, etc. This way we obtained the networks at class level where nodes are classes and edges are the mentioned relationship among classes (i.e inheritance, collaboration, etc.). Afterwards we annotated each class with the corresponding number of bugs retrieved using the procedure described in the following paragraph.

4.1 Retrieving Defectiveness Data

We considered the number of defects (bugs) as the main indicator of software quality. We collected data about the bugs of a software system by mining its associated Bug Tracking Systems (BTS). Bugzilla is the BTS adopted by Eclipse, where defects are tagged with a unique ID number. An entry in BTS is called with the common term 'Issue', and there is usually no information about classes associated to defects. Usually all the changes performed on the source code are reported on the SCM. To obtain a correct mapping between Issue(s) and the related Java classes, we analyzed the SCM log messages, to identify commits associated to maintenance operations where Issues are fixed.

We analyzed the text of commit messages, looking for Issue-IDs. Every positive integer number (including dates, release numbers, copyright updates, etc) might be a potential Issue-ID in the BTS. In order to avoid wrong mappings between a file and the corresponding Issue, we filtered out any number which did not refer to bug fixes. This operation was performed by associating Issue-IDs to files belonging to the same

release, and analyzing the commit logs to perform the mapping between Issues and classes. We associated to each release the Issues that are Bugs and that were classified as "closed" in BTS. In fact, very rarely Bugs which are labeled as "closed" are re-opened, this way being permanently associated with a release. The maintenance operations in Bugzilla are associated to files, called Compilation Units (CUs), which may contain one or more classes. Thus, in cases in which a file contained more than one class, we decided to assign all the defects to the biggest class of those Compilation Units. At the end of this process we obtained a network where to each node is associated the number of bugs of the corresponding class.

4.2 Metrics Analyzed

We computed the following metrics:

- *System Size*: the number of classes of the software system.

- *Average Bug Number (ABN)*: or bug density, namely the number of defects found in a system divided by the number of classes.

- *Modularity*: a measure of the strength of the obtained community structure, as defined in Section 3.

- *Number of Communities (NOC)*: the number of disjoint communities in which the network is partitioned.

- *Clustering Coefficient (CC)*: the average probability that if vertex i is connected to vertex j and vertex j to vertex k, then the vertex i will also be connected to vertex k. It can be defined as follows:

$$C_i = \frac{3 \times \text{number of triangles in the network}}{\text{number of connected triples of nodes}}, \tag{7}$$

  where a triangle is a set of three nodes all connected with each other, and a triple centered around node i is a set composed by two nodes connected to node i and the node i itself.
  The clustering coefficient for the whole graph is the average of the C_i's:

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i, \tag{8}$$

  where n is the number of nodes in the network [New03].

Release	2.1	3.0	3.1	3.2	3.3
Size	8257	11406	13413	16013	17517
Sub-Projects n.	49	66	70	86	104
N. of defects	47788	59804	69900	80149	95337

Table 1: Main features of the analyzed releases of Eclipse: size (number of classes), number of sub-projects (sub-networks), and total number of defects.

4.3 Dataset and Methodology

We analyzed 5 releases of Eclipse, whose main features are presented in Table 1.

Each release is structured in almost independent sub-projects. The total number of sub-projects analyzed amounts at 375, with more than 60000 nodes (classes) and more than 350000 defects.

We detected the modularity and its associated community structure for each subproject of each release using the Clauset-Moore-Newman (CMN) community detection algorithm devised by Clauset et al. [CNM04]. The latter is an agglomerative clustering algorithm that performs a greedy optimization of the modularity. The community structure retrieved corresponds to the maximum value of the modularity. Moreover, we retrieved the number of communities in which the networks are structured, the corresponding maximum value of the modularity, and the nodes associated to each community. The CMN algorithm implementation used is that provided by the R package *igraph* [CN06].

We then performed a correlation analysis among the network metrics and the software metrics (size and defectiveness) for each release on its own and also for the entire dataset, in order to have relevant statistics. Finally, in order to investigate the system evolution, we studied the relationship between network metrics and software defectiveness by cumulating the first and the second releases in a single set, then adding the third release to this first set to obtain a second set and so on. Specifically, we evaluated if, with a starting dataset of N releases, the best fitting curve for the cumulated $N-1$ releases could also be a good fit for the Nth release. To measure the forecast accuracy we adopted a χ^2 test. This way we were able to make predictions about the next release starting from those cumulated in the previous assembly.

5 Results

We performed different analyses among the network metrics and the software metrics for each release and the entire dataset. First and foremost, we noted a saturation effect of the number of defects and the clustering coefficient as the size of the analyzed systems increases. Our results show a general tendency for certain metrics to converge to a narrow range of values

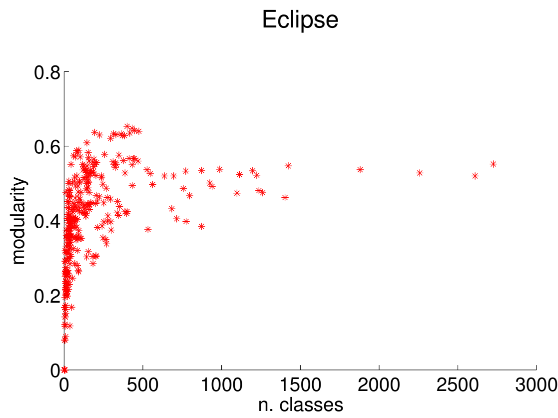


Figure 1: Scatterplot of the modularity vs system's size (n. of classes) for all subprojects

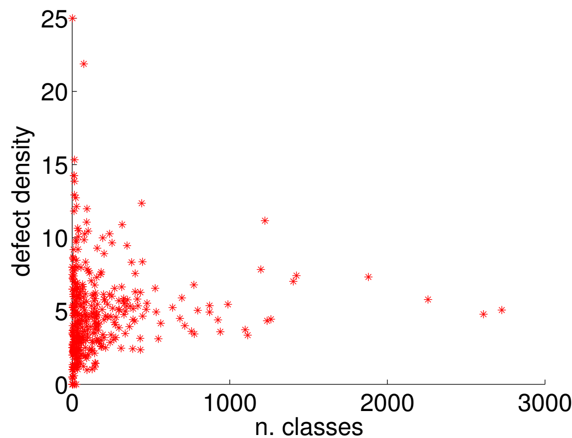


Figure 2: Scatterplot of the defect density (or ABN) vs system's size (n. of classes) for all subprojects.

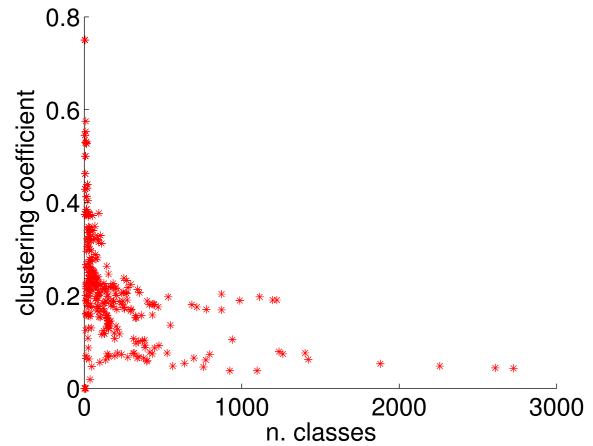


Figure 3: Scatterplot of the clustering coefficient vs system's size (n. of classes) for all subprojects.

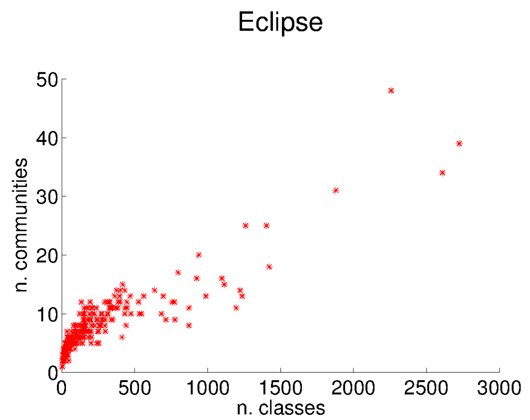


Figure 4: Scatterplots reporting the number of communities vs the sizes of the subprojects (n. of classes)

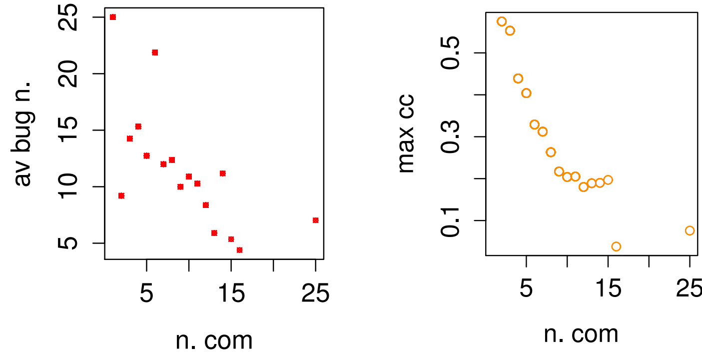
when the number of classes increases.

Figures 1, 2 and 3 show the relationship between systems' size (number of classes) and, respectively, modularity, average bug number (ABN) and clustering coefficient (CC).

All the metrics display more or less the same behavior. For relatively small systems, where the number of classes is roughly below 100, the metrics assume values in a wide range. Specifically, the defect density (or ABN) ranges from 0 up to 25, the clustering coefficient and the modularity, whose maximum value may be 1, range from 0 to 0.6-0.7. For system's size between 100 and 500 roughly, the variation ranges become smaller: the ABN lays between 2 and 12, the clustering coefficient lays between 0.05 and 0.2, and the modularity between 0.3 and 0.6. Finally, for fairly large systems, where the number of classes is above 500 or more, the metrics stabilize, showing small oscillations and eventually converging asymptotically to precise values.

Another interesting result is the monotonic increase of the NOC metric with system's size, reported in Figure 4. After a first nonlinear behavior the curve is aligned along a straight line. Additionally, our results show a significant correlation between NOC and both ABN and CC. Figure 5 displays a non-linear decay of the maximum values of ABN and CC versus NOC. It is worth to point out that for other network metrics, such as the mean degree or the average path length, there is not a similar trend.

In particular, Figures 5a and 5b show respectively the distributions of the maximum values of ABN and CC with respect to NOC for all the sub-projects for each release. Each point corresponds to the maximum value of the corresponding metrics computed on all the projects with the same number of communities.



(a) Average Bug Number vs. Number of Communities

(b) Clustering Coefficient vs. Number of Communities

Figure 5: Scatterplots of the relationships between the studied metrics.

Releases	α	r	χ^2	dof
2.1 - 3-0	-0.436	-0.920	0.065	16
2.1 - 3.1	-0.393	-0.934	0.059	17
2.1 - 3.2	-0.4302	-0.911	0.056	20
2.1 - 3.3	-0.404	-0.921	0.052	21

Table 2: Fit data for the power law between maximum ABN vs NOC: exponent α, correlation coefficient (r), χ^2 and number of degrees of freedom (dof).

Releases	α	r	χ^2	dof
2.1 - 3-0	-1.010	-0.654	0.075	16
2.1 - 3.1	- 0.917	-0.667	0.057	17
2.1 - 3.2	-0.977	-0.715	0.087	20
2.1 - 3.3	-0.986	-0.712	0.119	21

Table 3: Fit data for the power law between maximum CC vs NOC: exponent α, correlation coefficient (r), χ^2 and number of degrees of freedom (dof).

As these Figures illustrate, these values seem to follow a power-law like trend.

The distributions of the maximum values, when analyzed using a log-log scale, are well fitted by a straight line, suggesting two power-law-like relationships for the maximum values of both ABN and CC versus the number of communities, provided that the systems have the same number of communities. We applied a power-law best fitting algorithm in order to check this hypothesis, finding acceptable best fittings for the maximum values of CC and for the maximum values of ABN versus the number of communities. The power law parameters are reported in Tables 2 - 3. Table 4 shows the best fitting results, reporting the degrees of freedom and the normalized χ^2 for the relationship between these two metrics.

Eclipse	max ABN vs NOC	max CC vs NOC
dof	13	13
χ^2 / dof	0.361	1.005

Table 4: Fit data for the power laws between the maximum average defect density (max ABN) versus the number of communities and maximum clustering coefficient (max CC) versus the number of communities: correlation coefficient (r), normalized χ^2 and number of degrees of freedom (dof).

6 Discussion

We analyzed a large software system, Eclipse, using complex network theory with the aim of achieving a better understanding of software properties by mean of the associated software network. The application of the CMN algorithm confirms that the analyzed software networks present a meaningful community structure [ŠŽBB15, CMOT13]. Furthermore, the results show the existence of meaningful relationships between software quality, represented by the average bug number (ABN), and community metrics, in particular the number of communities (NOC) and clustering coefficient (CC).

The presence of a strong community structure in a software system reflects a strong organization of classes in groups where the number of dependencies among classes belonging to the same community (inter-dependencies) is higher with respect to the number of dependences among classes belonging to different communities (external-dependencies).

From a software engineering perspective this goal might be achieved by adopting good programming practices, where class responsabilities are well defined, classes are strongly interconnected in groups, and cou-

pling among groups is kept low. Within this perspective the network modularity can be seen as a proxy for software modularity.

Figure 1 shows that, with the exception of subprojects with less than 500 classes, the modularity does not increase along with the size, converging to values that range from 0.6 to 0.7. As reported in Section 3, these values indicate that the community structure is significant and well defined. At the same time Figure 4 shows that there is a linear relationship between the number of communities and the number of classes.

Such relationship is not trivial: the modularity and the number of communities are theoretically independent by the size [GMC10] and, in general, the number of communities does not increase with network's size. Moreover, by and large, there may be large networks divided in a small number of communities, depending on the network's topology. As a consequence our findings suggest that, in the examined case, it is possible to partition the software networks into a set of communities, where the number of communities is correlated with system's size.

Figures 2 and 3 report, respectively, the relationship of ABN and CC with the number of communities. Both metrics have a similar trend, with values converging to a range between 4 and 12 for ABN and between 0.2 and 0.6 for CC. This means that when the system's sizes increases the number of defects stabilizes and the same happens to the clustering coefficient. We already mentioned the significant increment of the number of communities with system's size. Since the increment of NOC is not trivial, this led us to assume that there might be a relationship among the topology of software networks, that determines the number of communities, and the other metrics.

Figures 5a and 5b show the distributions of the maximum values of CC and ABN for the projects with the same number of communities. As previously mentioned, this relationship seems to follow a power-law trend. The power law relating the NOC and the maximum values of ABN indicates that the community metrics, specifically the number of communities, can be exploited in order to evaluate the evolution of the defectiveness of a software system. In other words, once the relationship between NOC and the maximum values for ABN is known, one can evaluate approximately the maximum ABN in a future release of the same system, by computing the number of communities for that release.

This way, we might assume that systems with the same number of communities should have a number of defects per class lower than a given value. The same argument applies to the clustering coefficient of systems having the same number of communities. The relationship between CC and NOC is again a power law. This implies that if the NOC of an initial release (or of a set of releases) is known, one can in principle predict that in the following releases the clustering coefficient will not be greater than a certain value.

Releases	r	χ^2	dof
2.1 - 3-0	0.565	0.633	16
2.1 - 3.1	0.576	0.651	17
2.1 - 3.2	0.677	0.523	20
2.1 - 3.3	0.687	0.547	21

Table 5: Fit data for the maximum ABN vs maximum CC: correlation coefficient (r), normalized Chi squared (χ^2), and number of degrees of freedom (dof).

Figures 6 and 7 show, in a log-log scatterplot, the best fitting lines for the data discussed above. Each color corresponds to one set of releases cumulated according to the chronological order. The Figures confirm that the power-law like relationship appears in every cumulated release and is a regular and stable behavior throughout software evolution.

The power law parameters for the mentioned metrics and for each cumulated releases (see Section 4) are reported in Tables 2 and 3. As we can see they do not change significantly from one cumulated release to another. This suggests the existence of a progressively more stable behavior during software evolution, where the fitting with a power law becomes more accurate and tends to a fixed value as new releases are added in the cumulated dataset. These results might help developers to estimate the expected maximum ABN for software systems with a known community partition.

The two power laws indirectly connect the maximum values of ABN with the maximum values of CC in systems having the same number of communities. Such relationship can be made explicit reporting directly the scatter plot of the two metrics where each metric is computed for the same number of communities. Such plots are reported in Figure 6 for all the cumulated releases, and show that the two metrics are linearly correlated. Table 5 reports the correlation coefficient as well as the degrees of freedom and the χ^2 for such data for all cumulated releases. It shows that the correlation coefficient increases and the χ^2 decreases as new releases are added in the cumulated data, indicating a more stable relationship among the two metrics as the system evolves.

These results can be explained by noting that the larger the clustering coefficient, the higher is the number of classes linked to each other and the higher the probability of diffusion of defects among them. The topology of a software network is characterized by hubs, and the clustering coefficient in the area of the graph around any hub is higher by definition. If one hub is affected by one or more defects, it is more likely

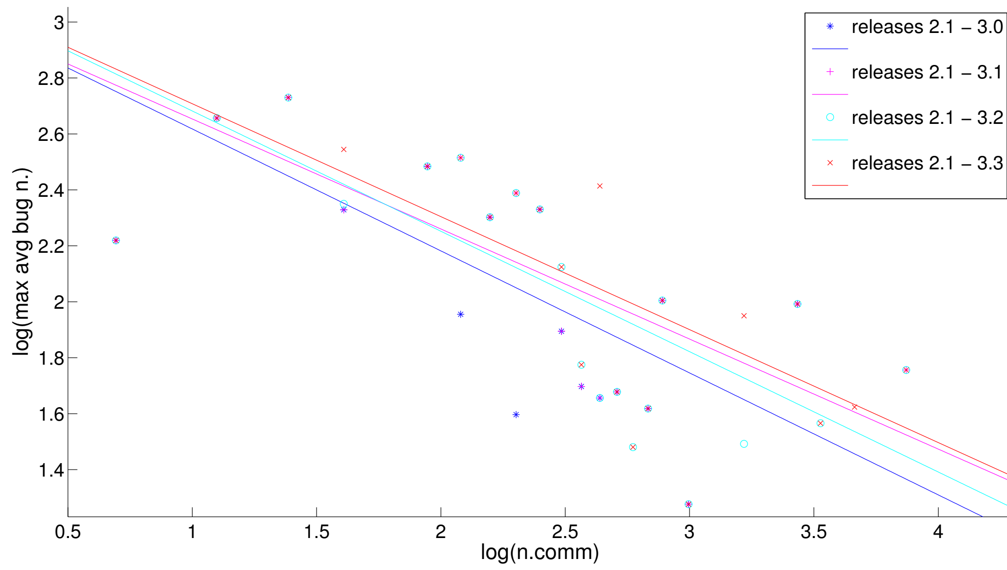


Figure 6: Cumulated log-log plots and best fitting lines of maximum ABN vs number of communities

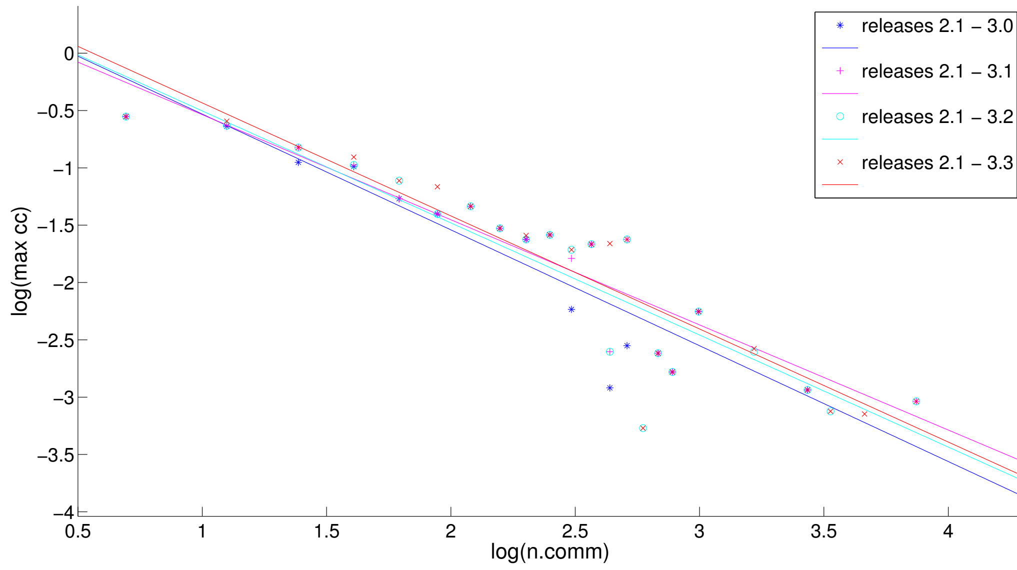


Figure 7: Cumulated log-log plots and best fitting lines of maximum CC vs number of communities

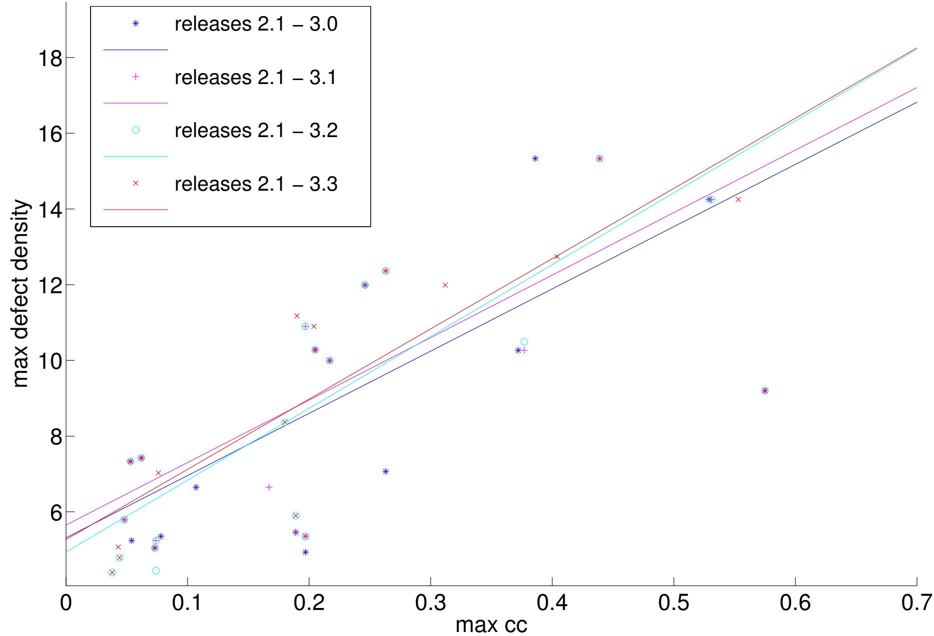


Figure 8: Cumulated plots and fitting lines for the maximum ABN vs maximum CC.

that these defects will spread among the classes connected to the hub, increasing the defect density in the area around it. This would explain the correlation among defect density and clustering coefficient.

7 Threats to Validity

This work suffers from some threats to validity, that are reported in the following according to the usual distinction in internal, external and construct threats to validity.

Internal Validity We conclude that the relationship among software defectiveness and community metrics can indicate that a high level of network modularity is related to low defectiveness, thus suggesting good programming practices. However, the relationships we found could be due to other phenomena, or deserve to be further investigated. What we propose here is just one possible formal explanation of our results.

External Validity We only consider one Java system, Eclipse, and analyzed its evolution. Our results should be validated on other systems and made more general. We are currently extending the analysis to many releases of NetBeans.

Construct Validity The rules reported in Section 5 might be faulty in some cases, not being able to correctly map defects to CUs [AMADP07]. There might be a bias caused by the lack of information on SCM [AMADP07], and there could be a chance of wrong assignments to happen for some classes. However, we considered the defect density as a relevant metric and studied its statistical distribution, which could be severely biased only in the case of systematic errors, which is not our case. Moreover, we consider only undirected edges in the class dependency graph, without making any distinction among the different relationships (i.e. inheritance, collaboration, etc.) that take place among classes. Consequently, our results could be refined considering the mentioned differences between the relationships and taking into account directed as well as weighted edges.

8 Conclusions

In this work we presented a longitudinal analysis on the evolution of a large software system with a focus on software defectiveness. We used a complex network approach to study the structure of the system and its modularity by computing the community structure of the associated network. After having retrieved the number of defects and associated them to the software network classes, we performed a topological analysis of the system defectiveness. We found a power law relationship between the maximum values of the clustering coefficient, the average bug number and

the division in communities of the software network. This led to a linear relationship between the maximum values of the clustering coefficient and of the average bug number. We showed that such relationship can in principle be used as a predictor for the maximum value of the average bug number in future releases.

References

[AMADP07] K. Ayari, P. Meshkinfam, G. Antoniol, and M. Di Penta. Threats on building models from cvs and bugzilla repositories: the mozilla case study. In *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, CASCON '07, pages 215–228, New York, NY, USA, 2007. ACM.

[ASTC11] Mahir Arzoky, Stephen Swift, Allan Tucker, and James Cain. Munch: An efficient modularisation strategy to assess the degree of refactoring on sequential source code checkings. In *ICST Workshops*, pages 422–429. IEEE Computer Society, 2011.

[BAJ00] A. Barabasi, R. Albert, and H. Jeong. Scale-free characteristics of random networks: the topology of the world wide web. *Physica A: Statistical Mechanics and its Applications*, 281:69–77, 2000.

[BC99] Carliss Y. Baldwin and Kim B. Clark. *Design Rules: The Power of Modularity Volume 1*. MIT Press, Cambridge, MA, USA, 1999.

[CK94] S. Chidamber and C. Kemerer. A metrics suite for object-oriented design. *IEEE Trans. Software Eng.*, 20(6):476–493, June 1994.

[CL04] D. Challet and A. Lombardoni. Bug propagation and debugging in asymmetric software structures. *pre*, 70(4):046109, October 2004.

[CMM+11] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu. On the distribution of bugs in the eclipse system. *IEEE Trans. Software Eng.*, 37(6):872–877, 2011.

[CMOT13] Giulio Concas, Cristina Monni, Matteo Orrù, and Roberto Tonelli. A study of the community structure of a complex software network. In *Proceedings of the 2013 ICSE Workshop on Emerging Trends in Software Metrics*, WETSoM '13, pages 14 – 20, New York, NY, USA, 2013. ACM.

[CN06] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[CNM04] Aaron Clauset, M. E. J. Newman, , and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, pages 1– 6, 2004.

[CS05] Hernn A. Makse Chaoming Song, Shlomo Havlin. Self-similarity of complex networks. *Nature*, 433(4):392–395, January 2005.

[DLR10] Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 31–41. IEEE, 2010.

[FB07] S. Fortunato and M. Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36, 2007.

[FMS01] Sergio Focardi, Michele Marchesi, and Giancarlo Succi. *A stochastic model of software maintenance and its implications on extreme programming processes*, pages 191–206. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[For10] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.

[GMC10] B.H. Good, Y.A. De Montjoye, and A. Clauset. Performance of modularity maximization in practical contexts. *Physical Review E*, 81(4):046106, 2010.

[GN01] M Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. U. S. A.*, 99(cond-mat/0112110):8271–8276. 8 p, Dec 2001.

[HBB+12] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, 38(6):1276–1304, 2012.

[MM07] Brian S. Mitchell and Spiros Mancoridis. On the evaluation of the bunch search-based software modularization algorithm. *Soft Comput.*, 12(1):77–93, August 2007.

[MMCG99] Spiros Mancoridis, Brian S. Mitchell, Yih-Farn Chen, and Emden R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *ICSM*, pages 50–. IEEE Computer Society, 1999.

[MTM+12] A. Murgia, R. Tonelli, M. Marchesi, G. Concas, S. Counsell, J. McFall, and S. Swift. Refactoring and its relationship with fan-in and fan-out: An empirical study. In *Proceedings of Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, CSMR '12, pages 63–72, 2012.

[Mye03] Christopher R. Myers. Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs. *Physical Review E*, 68(4):046116, Oct 2003.

[NAH10] Thanh HD Nguyen, Bram Adams, and Ahmed E Hassan. Studying the impact of dependency network measures on software quality. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[New03] M. E. J. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[New04] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69:066133, Jun 2004.

[New06] M. E. J. Newman. Modularity and community structure in networks. Technical Report physics/0602124, Feb 2006.

[NG04] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113), 2004.

[OWB05] Thomas J Ostrand, Elaine J Weyuker, and Robert M Bell. Predicting the location and number of faults in large software systems. *Software Engineering, IEEE Transactions on*, 31(4):340–355, 2005.

[Par72] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972.

[RS02] Valverde S. Cancho R. and V. Sole. Scale free networks from optimal design. *Europhysics Letters*, 60, 2002.

[ŠB11] L. Šubelj and M. Bajec. Community structure of complex software systems: Analysis and applications. *Physica A: Statistical Mechanics and its Applications*, 390:2968–2975, August 2011.

[SM96] Ron Sanchez and Joseph T. Mahoney. Modularity, flexibility, and knowledge management in product and organization design. *Strategic Management Journal*, 17:pp. 63–76, 1996.

[ŠŽBB15] L. Šubelj, S. Žitnik, N. Blagus, and M. Bajec. Node mixing and group structure of complex software networks. *ArXiv e-prints*, February 2015.

[TCM+11] 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. *Information Sciences*, 181(4):883–902, February 2011.

[TMT12] I. Turnu, M. Marchesi, and R. Tonelli. Entropy of the degree distribution and object-oriented software quality. In *Proceedings of the 2012 3rd International Workshop on Emerging Trends in Software Metrics (WETSoM)*, pages 77–82, June 2012.

[VS03] Sergi Valverde and Ricard V. Sole. Hierarchical small worlds in software architecture. arXiv:cond-mat/0307278v2, 2003.

[WKD07] Lian Wen, Diana Kirk, and R. Geoff Dromey. Software systems as complex networks. In Du Zhang, Yingxu Wang, and Witold Kinsner, editors, *IEEE ICCI*, pages 106–115. IEEE, 2007.

[Zha09] Hongyu Zhang. An investigation of the relationships between lines of code and defects. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 274–283. IEEE, 2009.

[ZN08] Thomas Zimmermann and Nachiappan Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pages 531–540, New York, NY, USA, 2008. ACM.