Predicting Software Quality through Network Analysis

Giulio Concas
concas@diee.unica.it

Michele Marchesi
michele@diee.unica.it

Cristina Monni
cristina.monni@diee.unica.it

Matteo Orrú
matteo.orru@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)

Abstract

We used a complex network approach to study the evolution of a large software system, Eclipse, with the aim of statistically characterize 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 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.

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 software systems quality and their modular structure. Depending on the fact that software systems are inherently complex, the best model to represent them is by retrieving their associated networks and the related topological properties [Mye03, ŠB, WKD07, ŠŽBB0, ZN08]. In a software network nodes are associated to software modules (e.g. classes) and edges are associated to connection between software modules (e.g. inheritance, collaboration relationships). We investigated the software modular structure - and its impact in software quality - by studying specific network properties: community structure, modularity and clustering coefficient. A community inside a network is a sub-network of densely connected nodes when compared to nodes outside the community [GN01]. Modularity is a function that measures how marked is a community structure (namely the way the nodes are arranged in communities) [NG04]. The clustering coefficient represents a measure of how much nodes are connected inside a network [New03].

We studied several releases of a large software system, Eclipse, performing a longitudinal analysis on the relationship between community structure, clustering coefficient and software quality. Our aim is to figure out if the studied metrics can be used to predict software quality of future releases.

This paper is organized as follows. In Section 2 we illustrate the methodology. In Section 3, we present and discuss some of our results and finally, in Section 4, we report our conclusions.

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

2 Methodology

In this work we aim at analysing the structure of a software system using its associated software network. In order to build the associated software network we parsed software's source code, retrieved from the corresponding Software Control Managers (SCM). During this procedure, we associate network nodes to classes and network edges to the various relationships between classes (inheritance, composition, etc.). We consider the number of defects (bugs) as a main indicator of software quality. To exploit this we collected data about the bugs of a software system by mining its Bug Tracking Systems (BTS). Bugzilla is the BTS adopted by Eclipse, where defects are tagged with a unique ID number. Usually 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 maintenance operations on software systems are reported on Software Configuration Management (SCM) systems like Concurrent Version Systems[1], Git and Subversion. However, it is not possible to distinguish between bug fixings or other actions such as enhancements, since all maintenance operations on software systems are recorded as "commit operations". To obtain a correct mapping between Issue(s) and the related Java files (CUs), we analyzed the SCM log messages, to identify commits associated to maintenance operations where Issues are fixed. If a maintenance operation is done on a file to address a defect, we consider the CU as affected by that defect.
We first analyzed the text of commit messages, looking for Issue-IDs. Unfortunately, every positive integer number is a potential Issue-ID. However, sometimes numbers that refer to maintenance operations are not related to Issue-ID resolution, but, for example, to branching, data, number of release, copyright updating, and so on. To avoid wrong mappings between Issue-IDs and CUs, we applied the following rules:

- In each release, a CU can be affected only by Issues which are referred to in the BTS as belonging to the same release.
- All IDs not filtered out are considered Issues and associated to the addition or modification of one or more CUs, as reported in the commit logs.
- When assigning defects to classes in the corresponding CUs, since there were few CUs containing more than one class, we decided to assign all the defects to the biggest class of those CUs.

This method might not completely address the problems of the mapping between defects and CUs [AMADP07]. In any case we checked manually 10% of randomly chosen CU-defect(s) associations for intermediate releases and every CU-defect association for 3 sub-projects without finding any error. A bias may still remain due to lack of information on SCM [AMADP07]. The subset of Issues satisfying the conditions as in Eaddy et al. is the Bug-metric [EZS+08]. Of course there are chances for wrong assignments to happen for some classes, but since the average number of defects for class is very low, the number of wrong assignments in the entire system, considering also CUs with one class, is very limited.
At the end of this process we obtained a network where to each node is associated the number of bugs of the corresponding class.

We collected the source code and analysed 5 releases of Eclipse, whose main feature are presented in Table 1.

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 analysed releases of Eclipse (EC): size (number of classes), number of sub-projects (sub-networks), and total number of defects.

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

We detected the associated community structure using the algorithm devised by Clauset et al. [CNM04]. This is an agglomerative clustering algorithm that performs a greedy optimization of the Modularity (Q) [New03]. At the end we retrieved the number of communities in which the network is structured, the corresponding value Q of the modularity, and the nodes associated to each community. We performed the computation of the clustering coefficient using the implementation included in the IGraph software [GC06]. To study the system's evolution we used the following approach. We first carried out the analysis for each release, and then we assembled together different releases, according to a temporal evolution. More precisely, for the 5 releases of our dataset, we studied the evolution of the system 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. This way we were able to make predictions about the next release starting from those cumulated in the previous assembly.

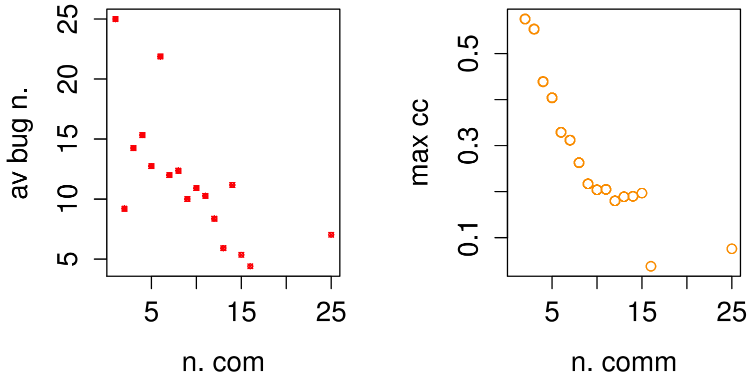
[1] CVS, http://www.nongnu.org/cvs/.

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 2: Results on the power law between maximum Clustering Coefficient vs Number of communities for Eclipse: exponent α, correlation coefficient (r), value of Chi Squared (χ^2) and number of degrees of freedom (dof).

3 Results

Figures 1a and 1b show the distributions of the average bugs number (ABN, Fig. 1a) and of clustering coefficient (CC, Fig. 1b) with respect to the number of communities (NOC) for all the sub-projects of all the releases. Although the scatterplots for the relationship between NOC and other metrics are sparse, the reported scatterplots show the existence of a power-law-like relationship between the maximum values of the mentioned metrics. This led us to hypothesize a linear relationship between the maximum values of CC and the ABN. In Tab. 2, on the left, the power law exponents, the correlation coefficient, the χ^2 and the degrees of freedom (dof) for the best fitting in log-log scale are reported. They refer to different "cumulated" releases for the relationship between CC and NOC. Table 2 shows how the power laws parameters do not change significantly from one cumulated release to another. This suggests the existence of a progressively more stable behaviour 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. The same considerations can be applied to the relationship between maximum ABN and NOC. The scatterplot portraied in Fig. 2 shows the relationship between the maximum defect density versus the maximum clustering coefficient, for all the cumulated releases, along with the best fitting straight line. We investigated if, starting with a dataset of N releases, the best fitting curve for the cumulated $N-1$ releases could also be a good fit for the Nth release. In order to measure the forecast accuracy we adopted a χ^2 test. Table 4 reports the results of the best fitting for the relationship between CC and ABN showing that the linear correlation is not very high. Nonetheless, the χ^2 test returns an high level of significance. Table 2 reports the results of the analysis on the forecast for software quality. We computed the ratio between the χ^2 and the degrees of freedom. According to the results reported on Table 2, on the right, the χ^2 values are close to 1, meaning that for the given degrees of freedom the fits are good.



(a) Average Bug Number vs. Number of Communities (b) Clustering Coefficient vs. Nu of Communities

Figure 1: Scatterplot of the relationships between the studied metrics.

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

Table 3: Fit data for the power laws between the maximum average defect density (max ADD) versus the number of communities and maximum clustering coefficient (max CC) versus the number of communities: correlation coefficient (r), normalized Chi squared (χ^2), and number of degrees of freedom (dof).

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 4: Fit data for the maximum defect density vs maximum clustering coefficient: correlation coefficient (r), normalized Chi squared (χ^2), and number of degrees of freedom (dof).

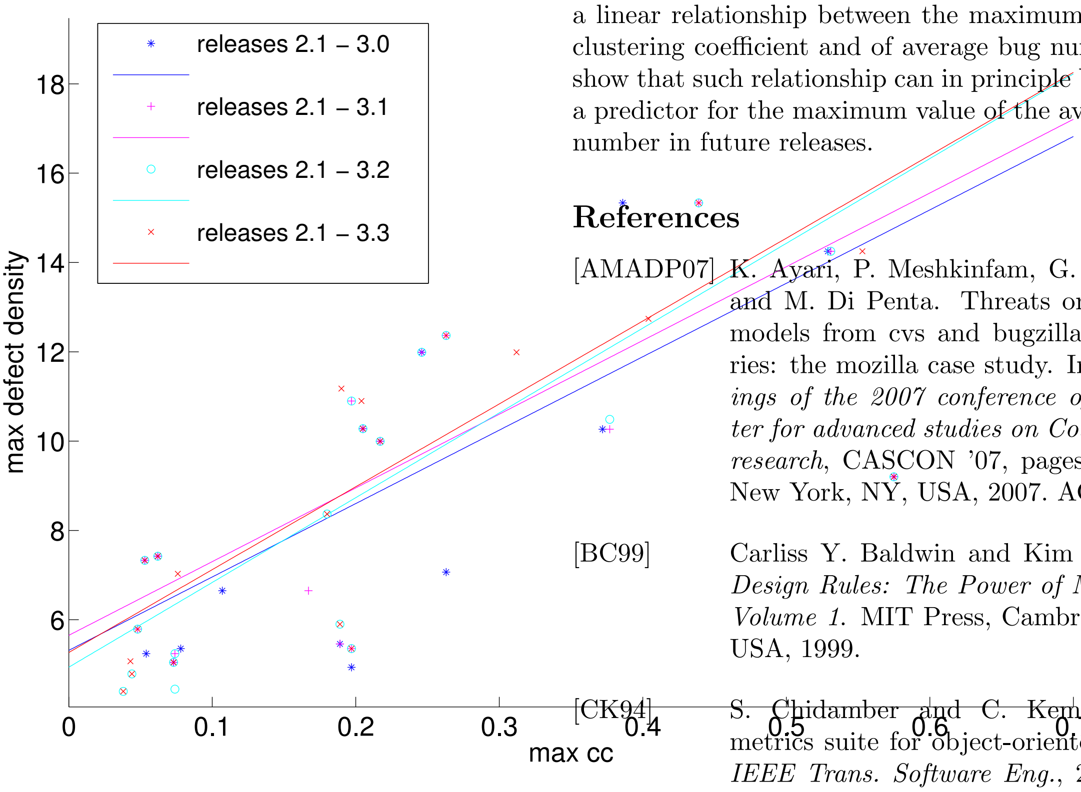


Figure 2: Cumulated plots and fitting lines for the maximum defect density vs maximum clustering coefficient.

4 Conclusions

In this work we presented a longitudinal analysis on the evolution of a large software system with a focus on software defectiveness. Through a complex network approach we were able to study the structure of the system by retrieving the community structure of the associated network. After retrieving the number of defects associated to the software network classes, we performed a topological analysis detecting the community structure. 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 lead to a linear relationship between the maximum values of clustering coefficient and of average bug number. We show 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.

[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.

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

[EZS+08] M. Eaddy, T. Zimmermann, K.D. Sherwood, V. Garg, G.C. Murphy, and et al. Do crosscutting concerns cause defects? *IEEE Transactions on Software Engineering*, 34(4):497–515, November 2008.

[GC06] T. Nepusz G. Csardi. The igraph software package for complex network research. *InterJournal of Complex Systems*, page 1695, 2006.

[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.

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

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

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

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

[ŠB] L. Šubelj and M. Bajec.

[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.

[ŠŽBB0] L. Šubelj, S. Žitnik, N. Blagus, and M. Bajec. Node mixing and group structure of complex software networks. *Advances in Complex Systems*, 0(0):1450022, 0.

[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.

[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.