International Journal of Future Generation Communication and Networking
Vol.10, No.4 (2017), pp.109-118
http://dx.doi.org/10.14257/ijfgcn.2017.10.4.11

# Explore the Community Structure of the Software Network

Munawar Hussain[1], Ren JiaDong[1] and Awais Akram[1]

[1]*College of Information Science and Engineering,*
*Yanshan University*
*No. 438 Hebei Avenue, Qinhuangdao 066004, P.R. China*
*munawarjut@yahoo.com*

## *Abstract*

*Several complex networks have a natural community structure mixture of different types of nodes and edges. Each community, or division, densely connected with each other as subgroups. In terms of software systems, class interdependency networks with community structure are networks which characteristics of nodes or classes have not been investigated completely; it is very helpful for improving and evaluating software class structures. In this paper, we explore a software system of nodes and edges where nodes show as classes and the relationship between them as edges which is called a complex software network. To construct the complex software graph, we map the software classes and edges to graph by the source code. To get interesting results, we apply different metrics on complex software network and found that it has a great influences and interdependencies. Specifically, we found the communities number and size, in which a complex software network can be divisions and their modular. Moreover, this result is helpful to find software clustering fault proneness in term of associated software graph structure quantities related problem.*

***Keywords:*** *Complex software network, modularity, community structure, software bugs, graph structure, software metrics*

## 1. Introduction

Contemporary software networks are very complex and vast networks because it's made of thousands and millions of code line with multiplex structure. Since last decades, Object Oriented (OO) programming got much attention, because of its simplicity, and object oriented (OO) software networks used in a large number of computer applications. To build any kind of Software network, according to the approach of object oriented (OO) it has become quite much possible to establish varying kinds of software networks, where nodes shows as particular software modules and link between them shows as the relationship of all software modules.

Several software networks showed that they have the characteristics and properties of complex network [1-7]. During the last decade, complex network got much attention of scientist to investigate the several quantities, which has used, and define characterizing and measuring their nature of complexities. In terms of computing, such quantities large numbers of software programs and algorithms have been built. Complex network has many great features, one of them feature is easily partitioning of them into smaller groups or networks. Such sub networks and sub modules are known as communities, in which nodes are very densely connected with each other in a community, whereas with other community node's connections are very rare [8]. Because of this faith, software network is a magnificent, suitable for use complex network tool to apply community structure method on a software network to acquire communities of software modules or classes, because software modules and classes are organized in a hierarchical mode, where tiny components collaborate together. Software engineering, implementation stress on the

ISSN: 2233-7857

decay of multiplex jobs into small parts, to promote the development processes smoothly [9], and these software networks are much evolvable designed [10]. In spite of this, little work done by researchers to explore the community system of the complex software network. It revealed that software networks have a strong community structure which is not same to packages formation developed by the developer.

This paper will show the results of investigating the metrics which proved that it can be beneficial to mark the community structure of a software system which it can be easy for distinguish the possessions of the corresponding complex software network. We show results by using big, complex software network, which is NetBeans IDE 8.2, consists of 56 smaller subproject. First of all, we get source code of the open source software, NetBeans IDE 8.2 with the help of some tools on Linux, then we examined the source code which consists of more than 43000 classes, Later we examine the inter class dependencies in the form of constructing a complex software network. In complex software network nodes treated as classes and edge between them treated as relationship among classes. The result shows that 56 smaller subprojects do not depend on others and could be examine as different 56 software networks. By apply the community detection algorithm, we got different kind of communities of the NetBeans IDE 8.2 which examine as independent systems and at next step we computed some community metrics, like modularity, mean distance, clustering coefficient and mean node degree. To get a graphical representation of community structure, firstly we build static Graphs, where nodes represent as a software classes and edges as the relationship between the software classes or nodes. We discovered that, there is a very strong correlation among the modularity, mean degree, community metrics and average path length to software fault and software quality. Moreover, we performed an examination at global ranking, in term of correlation between number of packages and a number of communities. From the result, we explore that quantity of packages is methodically larger than the quantity of communities, which presenting that the partitioning apply by the developers is distinct from the result of community structure quantified by the algorithm. This result, most likely associated with projectile choice. Lastly, we explore the correlation between software faultless and communities which showed that if the numbers of communities are increasing then mean bug numbers also increased with it.

The rest of the paper is organized as follows in Section 2 we discuss the background of complex software system as a complex network with the significance of community detection. In Section 3 we will discuss the special methodology which applies for assignment to classes and bug extraction, and evaluate the associated metrics and community detection algorithm which we used to get different communities. In Section 4 will discuss and show the experiential finding, and in Section 5 we sum up all and present conclusion.

## 2. Related Work

Many software systems have reached such a huge dimension that it looks sensible to treat them as complex networks [1-2], [14]. In particular, software networks made of thousands of nodes show properties typical of complex networks [3], like a power law distribution of the node degree, scale free properties [4], fractal and self-similar features [26], the small world property, power law scaling for bug distribution [15], for refactored classes [16], and so on. In the field of software, class diagrams [17-18], the software collaboration graph [2], the package dependencies networks [19], have been found to belong to the complex networks category. In our study nodes represent as Net Beans classes and edges represent as class dependencies.

Pan *et al.*, [20] proposed exploiting community results to identify the refactoring point in the software evolution process by observing evolving trends of modularity [21] (Newman and Girvan, 2004), a metric which was originally proposed to measure the

 

quality or significance of a community structure. In [22] it has been shown that the knowledge of scale-free properties of software networks could be useful to reduce the time devoted to maintenance. Subelj and Bajek, after analyzing different Java software's using three community detection algorithms, find that the software systems analyzed exhibit a significant community structure that does not match the packages structure imposed by the developer [12]. Community structure is much usual in real world networks. Social networks include community groups (the origin of the term, in fact) based on common location, interests, occupation, *etc*. [23] Community structure is a very important feature of network theory, in recent years, which got much attention of scientists. Inside the community of networks, there are large numbers of vertices which are densely connected with each other through edges. The community structure of a network is its division in subgroups or communities [8]. The vertexes, which belonging to the same community, have more chance to have the same properties and behavior, or show a same behavior unit. Community detection is traditionally addressed using techniques like hierarchical clustering and partition clustering [24] and it faces different problems. Most important is to search a suitable algorithm that allows system scalability to do different types of order of immensity, which could be hundreds to billions of nodes in a system to be examined in a short time. Newman et al. Proposed several algorithms for community detection [8], [34], [25] using different approaches and dealing with the problem of the computational burden. Moreover, in [8] Newman and Girvan introduced a quality function called modularity, to evaluate how well a network partition in communities is. In our experiments, exploration of community structure of complex software system is very helpful to know how the complex software network is structured in different modules or classes. In our this paper, First we apply community detection algorithm for software static graph to get communities of the NetBeans IDE 8.2 software, After obtaining 56 potential communities, we examine the communities of 56 subproject and their community possessions of every subproject in term of recuperate information about the complex software network structure and their quality. We evaluate the three different types of metrics on our obtaining communities such as modularity, average path and mean degree, clustering coefficient and at least we differentiate metrics on data obtained from subproject, like the quantity of packages or quantity of defects.

## 3. Proposed Method

Our data set software is NetBeans IDE 8.2 which already has been explored in the research by another researcher. The data set which we use relate to a broad set of different networks, data collected by other authors and examine in previous works [13]. This software is large software with more than 4300 classes and 56 smaller subproject. In order to construct the complex software network, we analysis the software source code to search whether their classes have dependencies on each other or not. With the help of CMN algorithm we detected communities where graph nodes represent as classes and links between the nodes as a relationship. The examination of the code could be depending, mixture and inheritance. After having an examination of the source code for construction the associated software graph, we evaluated distinct software network metrics. At the next step, we obtained the classes, which influenced by bugs, from bug place and related the classes to corresponding classes. At last; we put the community examination process to acquire and measured different community metrics. By this way, we can obtain some positive signal that it can help to produce fault proneness of the software system.

### 3.1. Metrics Analyses

When building the complex software network, the aim of metrics examination is to explore the potential correlation and association between the properties of complex

software network and structure. In terms of complex software network, we compute the metrics such as what is the system size in respect of software system classes, and quantity of distinct sub-packaged used to place the class which used by designers. Another metric is bug quantity which exists in the software classes. The quantity of bugs exists in the software system if division them with the classes' numbers, we get mean bug numbers. In terms of community structure and complex software network, we analysis the following measurement of software network. Next metric is modularity which was modeled to compute the robustness of groups of networks into divisions called communities or clusters [27]. It calculates the condition of the community structure. For example, take a software system and dividing it into different communities, then define a matrix as c × c matrix e where it consists of $e_{ij}$ means it connects nodes in the community I to nodes in community j by matrix, we can obtain the fraction of links in software network connecting nodes in the identity community by calculating traces as Tr e= $\sum_i e_{ii.}$ If the computing result is near to 1 and has greater value, it's an evidence of obtaining an appropriate community structure, but this supposed to be the very insignifiant case. To obtain a potential quality measurement of community structure, the fraction is [8] compared with the expected value of the same fraction of a network with the same community divisions but random connections between the vertices. With respect the matrix, we define column total $a_i = \sum_j e_{ij}$ fractions of links, attach to nodes inside community i. To differentiate these values, we get the modularity by following subtraction formula.

$$\mathbf{Q} = \sum_{\boldsymbol{i}} \left( \boldsymbol{e_{ii} - e_i^2} \right) = \boldsymbol{Tr}\ \boldsymbol{e_{ij}} - \sum_{\boldsymbol{ij}} \left( \boldsymbol{e_{ij}^2} \right) \quad (\mathbf{1})$$

If the Q value is near to 1 it's an indication of a good quality community structure. Typically, founded value is between 0.36 to 0.74. Quantity of communities: The distinct communities in which a complex software network can be partitioned will be discussed later. The geodesic distanced between software network vertexes called average shortest path. Clustering coefficient metric uses for accounting the cluster tendency, which means if node A has link to node B and node B connected to node C thus, node A considers connected with node C too. It equation defined as [26]

$$\mathbf{C} = \frac{\mathbf{3} \times \mathbf{No\ of\ triangles\ in\ the\ network}}{\mathbf{No\ of\ the\ connected\ triples\ of\ vertics}} \quad (2)$$

In this equation there is a triangle which shows three vertices, all are related to with each other and number of related triples is the number of edges between the nearest neighbors of a node inside a clique [35]. If the C value is high its mean there is higher elastic of the removal of nodes, which has properties of a real world network.

### 3.2. Bug Tracking System

Normally this method used to locate bugs, feature them and enhancement of the complex software system. There are several bug tracking systems which apply by open source software projects where end users can get access directly to bug reports [28]. In our used open source NetBeans software BTS uses as issueillz. IssueZilla (IZ) is the issue (bug) and tasks management system used by Community to ensure the proper issues management and the completion of the undergoing tasks inside the Community. The Bug tracking system does not include data which concerning classes related to bugs [29]. While the common versioning system (CVS) of the software arrangement management system keeps all the data and track of arrangement operations [30]. In CVS all actions take down in CVS. For CVS it very impossible to inquiry about how many and what

 

types of actions already have been completed for fix bugs and an enhancement or present some new feature, because these all operation in CVS are not in a structured way. These all above mentioned actions are carried out on files, which known as Compilation unit (CUs) and could be consisted of a single class or more than one classes. To find the Issues impacting classes, we needed to match the information with saved information in software arrangement management system [30].

In order to get an exact mapping among the bugs and the associated object oriented files, we examine log messages of CVS, to find the commits related to preservation processes where bugs are going to be fixed. If the preservation processes are completed on a file to fix a bug, we contemplate the Compilation unit as influenced by the issue or bugs. At the next step, we allocate the bugs/issues to classes in the related Compilation unit (CUs), But there is very little Compilation unit (CUs) consists of multiple classes, so better to allocate all the issues to the larger class of Compilation unit. This technique may be not be completely addressed the complications of the mapping among the bugs and Compilation unit CUs [3]. In every time we examine the non-automatically 15% of Compilation unit issue related to every report and every Compilation unit bug/issue for four projects and found no mistake. A prejudice may be remaining because of loss of data on CVS [31]. The subparts of bugs fulfilling the states of the Bug-metric [32]. Noticeably, there are exists some chance for incorrect tasks occurring in some classes.

### 3.3. Community Structure

In our paper, we use the CMN algorithm for community detection [25]. It's very common used algorithm in the world by scientist because its efficiency and performance quite well. It's quite faster. CMN utilizes some shortcuts which are quite helpful to reduce the computational time to some extent. For example, a network with m edges and n nodes the computational time is $O(n\,log^2\,n)$. This method quantifies the dissimilarity in the modularity which acquired by mixing numbers of communities till just remain a last one community. It works with some special properties in a software network. The algorithm quantifies the varying in modularity got by connecting the communities till just single community is remaining. The clustering evaluate is quantified by special characteristics of the system. Its adjacency matrix as follows.

$$\mathbf{A_{vw}} = \begin{cases} \mathbf{1} & \textbf{If v and w are joind} \\ \mathbf{0} & \textbf{otherwise}, \end{cases} \tag{3}$$

In the equation 3, v and w are two different nodes, those owned by communities $c_v$ and $c_w$ and the degree of nodes and the quantity of there in-links:

$$\mathbf{K_v} = \sum_{\mathbf{w}} \mathbf{A_{vw}} \tag{4}$$

The modularity stated as in equation (1) can be shown as task of these properties, we know the fraction of relationship linked to nodes in *i* community to nodes in *j* community, we can write as follows:

$$\mathbf{e_{ij}} = \frac{\mathbf{1}}{\mathbf{2m}} = \sum_{\mathbf{vw}} \mathbf{A_{vw}}\ \boldsymbol{\delta}\left(\mathbf{c_{v,}i}\right)\boldsymbol{\delta}\left(\mathbf{c_{w,}j}\right) \tag{5}$$

Thus, vector$a_i$ , nodes related to links in a community i, is as following,

$$\boldsymbol{a_i} = \frac{\mathbf{1}}{\mathbf{2m}}\sum_{\boldsymbol{v}} \boldsymbol{k_v}\,\boldsymbol{\delta}(\boldsymbol{c_v}, \boldsymbol{i}) \tag{6}$$

In this algorithm, it considers each node like a community, and it starts to linked 2 communities, it quantifies the varying in modularity, it obtains higher values, and starts the related joining, till just a single community is rest. Thus, it starts as,

$$\Delta Q_{ij} = \begin{cases} \frac{1}{2m} - \frac{k_i k_j}{(2m)^2} & \text{if i and j are joind} \\ 0 & \text{otherwise,} \end{cases} \quad (7)$$

For every set of communities, this algorithm evaluates a matrix consist of $\Delta Q_{ij,}$ and store the vector $a_i$ and its maximum -heap H values with biggest components of labels i, j and $\Delta Q_{ij.}$ As soon as the biggest modularity value is chosen, the algorithm acting the corresponding linking the two different communities, as ΔQ increases Q also increase, and this approach repeats till remain just single community.

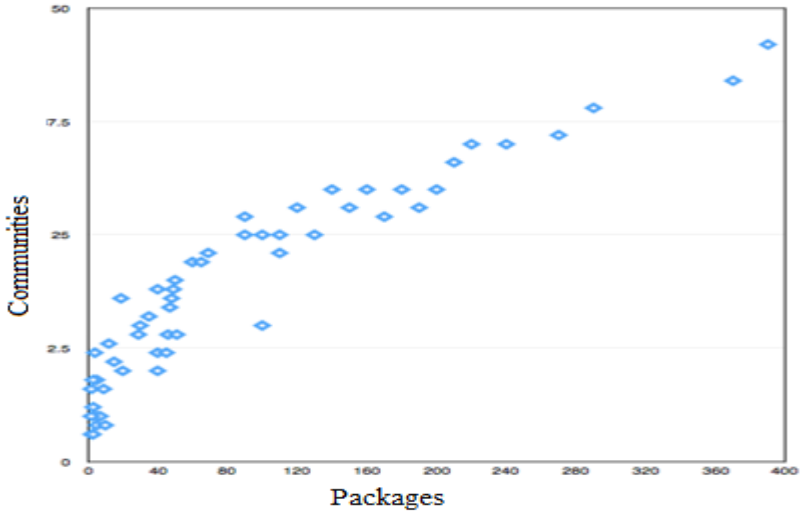


**Figure 1. Communities' vs Packages**

## 4. Experiment Analysis

In this part we will examine the results of experiments. Firstly, we show the global outcome acquired as taking the collection of 56 sub projects. Then we analysis the lack of consistency in each subproject. In the figure 1, it shows the scatter plot of comparison of software packages and communities for 56 subprojects. This plot presents a great correlation between variables and then it shows the sub linear communities number dependencies on the package numbers, but it's not happening in tiny projects, Here Pearson correlation is 0.9221 and significance which also called (P-value) is about 0. By such correlation values, we can judge because these two variables are linked to software system size, but on the other hand, sub linear dependence presents that division in packages set by the developer is good than the division throughout the communities got with community detection algorithm, which considers on the edges between vertexes .If the developer puts the classes with the identical aims and functionality in complex software network packages, then we come to know that community structure recommend alternately that among the software packages are much interdependencies, so they cannot seen as isolate software communities on the basis of connections. If the classes of the software system are less than 100 then outcome is not true. But if packages numbers are bigger than the community's number, its mean that single packages can be divided into communities. To make it more interesting, we compare the result of class numbers with

 

the dependence of packages numbers, which is very nearly excellently linear (Figure 2) with linear correlation ρ = 0.9458 and p-value is about 0. It shows that a developer wants to build packages with approximately the same class numbers, so classes mean number per package stay unvarying. But for a number of communities, it's not right, where classes mean community per community expands with the software system size.

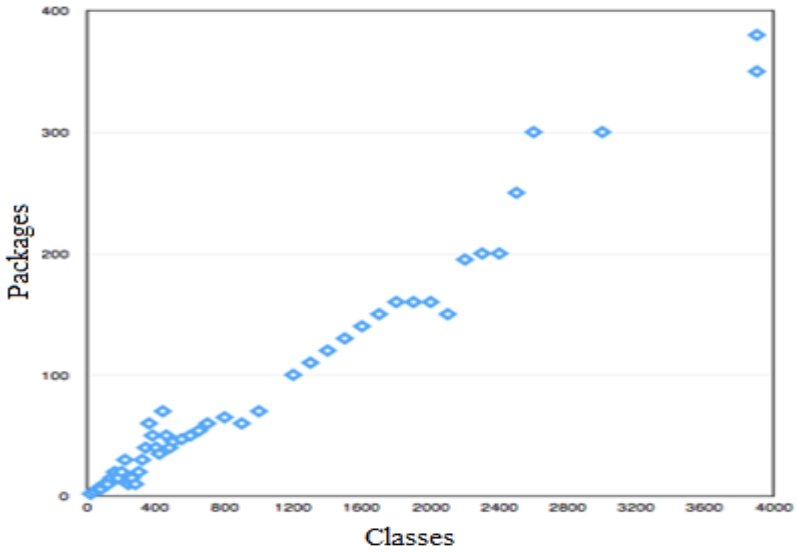


**Figure 2. Packages Vs Classes**

**Table 1. Correlation Coefficient and P-values between Communities' Numbers and Bugs Number of Whole Software System and Average Size Software System between 250-1000 Classes**

| | Whole System | Average System |
|---|---|---|
| Correlation Coefficient | 0.1150 | 0.6640 |
| P-Value | 0.4270 | 0.0050 |

The above experimental data as given in table, presents if the system is bigger, there are links between the nodes are they densely connected with each other. If looks it as the software system, point of view, it tells us if the software system bigger than the class groups does not depend on each other, also bigger. At the next step we take account the system bug numbers, which metric size does not depend, however it's a system quality signal. The total bug number correlation with communities' numbers in every single subproject is very inferior in total. On the other hand, when we moderate examine to average systems, from rage 250-1000 classes, that shows expand the correlation with communities, number. In respect of software system quality, this correlation shows if the detected communities are bigger in numbers than software system quality is poor. Table 1 presents the p-values and correlation coefficient between the all system communities and average size system communities for 56 systems and limit of classes from 250-1000.In the respect of software development, all mentioned different condition such an explanation. In reality, the variation comes handling the average system, but for tiny and bigger systems this variation does not come. From the small system point of view, statistics are very less. Actually, if the classes of a system are less than hundreds then it's quite difficult to partition them into communities.

For extremely big system, statistics are also very big, and other technique can apply. There is one possible way is to do the division in several communities indicates that classes are organized in several tiny systems, where classes are much combined with existing system classes. Consequently, to acquire a bigger community's number, coupling within classes should be globally less and more locally. It shows that general classes incline to get higher values, and it has been indicated to have faults and bad prone [33].

**Table 2. Correlation between Different Metrics**

| Network | Clustering coefficient (CC) | Mean Degree (MD) | Average path length (APL) | Modularity (Mod) | Median (Med) | Bug percentage |
|---|---|---|---|---|---|---|
| Cnd | 0.540 | 0.790 | 0.655 | 0.574 | 0.325 | 99.5 |
| Core | -0.472 | 0.870 | 0.643 | 0.638 | 0.475 | 75.4 |
| Editor | 0.330 | 0.80 | 0.753 | 0.750 | 0.310 | 97.1 |
| Enterprise | 0.089 | 0.735 | 0.615 | 0.551 | 0.309 | 93.0 |
| J2ee | 0.499 | 0.821 | 0.784 | 0.763 | 0.276 | 92.8 |
| Mobility | -0.179 | 0.862 | 0.781 | 0.786 | 0.234 | 99.1 |
| Ruby | 0.709 | 0.713 | 0.570 | 0.509 | 0.311 | 90.4 |
| Server plugin | 0.599 | 0.831 | 0.811 | 0.788 | 0.222 | 95.0 |
| Uml | 0.609 | 0.788 | 0.688 | 0.674 | 0.194 | 94.7 |
| Visual web | 0.070 | 0.657 | 0.479 | 0.472 | 0.307 | 93.9 |
| Web | 0.586 | 0.560 | 0.582 | 0.634 | 0.352 | 79.4 |
| Web Svc | 0.579 | 0.767 | 0.697 | 0.728 | 0.367 | 95.9 |
| Xml | 0.615 | 0.734 | 0.653 | 0.650 | 0.320 | 93.9 |

Other side, if the software system is quite bigger than complexity grows and developer needs to slack plain power of the complex software network graph structure. Global coupling expands and communities' numbers also expand. So the correlation between communities' numbers and bug number vanishes. As the result presents high correlation between communities' size and bugs numbers. Here is a very interesting fact is, that linear correlation, not at peak, while are organized common connection between the community metrics and number of bugs, But there is a very interesting thing to see which is the linear correlation is not at top level, while many general connections between the community metrics and bug number, aside through the clustering coefficient. To measure and analysis the features we quantify the correlation coefficient and p-value for association between the bug numbers and inside the modularity, community and clustering coefficient. For correlation coefficient, it's very easy to find threshold influence.

In Table 2, we examine the modularity's medians for 13 systems, the proportion of issues/bugs related to the groups which are above the median, in principle to guess the threshold influence. Our result presents the correlation between the community metrics and number of bug is 0.55 to 0.84. Moreover, for most of the networks the proportion of issues/bugs related to the communities where the 80% modularity is higher than the median. The communities modularity medians are quite near to all networks, about 0.21

 

to 3.1, and one network showing 0.475.So if single select 0.3 as threshold value of the whole network, the proportion of bug/issues in the communities the threshold is near to 80%-90%. The number of bugs extends in every community with its size. To examine the size effect on correlations, we executed some text on many networks. Specifically, we examine if the network size larger whether the size of modularity also larger or not.

We quantify modularity and found that it cannot change at a big level, if the size of the network increases. By this example, it's showing that there is no common correlation between network size and community metrics. In respect of the complex software network, size and issue/bugs often have good correlation with each other; however, our results present modularity metrics are not larger for larger communities.

## 5. Conclusion

In our paper, we examine an object oriented software system NetBeans IDE 8.2 Java programming, which contains more than 4300 classes and 56 sub projects. To analysis it as a complex software network graph, we construct a software network including all subproject. To get different communities and modularity we apply CMN algorithm on a complex software network. Moreover, we quantified different community metrics like average length, mean degree and clustering coefficient for every network. Issuezilla is used as the bug repository to obtain the bug number and then link them to a number of bugs to classes of complex software network. We can quantify the community examination in order to acquire the detail about what the developer made the design of the classes. We found that the average size network has a community which shows linked to the bug class. So, for such networks, the division in bigger number of modularity of classes can have defectiveness. Eventually we come to know when quantified the mean degree, modularity and average length in a single system, showing a strong sign for bugs of software. We select the median modularity a threshold value to differentiate the bug defective communities and without bugs communities. We suppose this procedure is excellent to choose the fault proneness communities, such as through maintenance and test. In our future research work, we will include new metrics and new techniques to explore complex software system more vastly.

## Future Scope

This study would help the reader to understand the structure of classes and the solid modularity in software classes because of the relationship in the code. Furthermore, by using software metrics, it would be easy to find software bugs and bugs related carried modules during the software preserving activities.

## References

[1] S. Focardi, M. Marchesi and G. Succi, "A stochastic model of software maintenance and its Implications on extreme programming processes", Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, **(2001)**, pp. 191-206.
[2] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of Software collaboration graphs", Phys. Rev. E, vol. 68, no. 4, 046116, **(2003)** October.
[3] A. Barabasi, R. Albert and H. Jeong, "Scale-free characteristics of random networks: the World wide web. Phys", A, vol. 281, **(2000)**, pp. 69-77.
[4] H. A. M. Chaoming Song and S. Havlin, "Self-similarity of complex networks", Nature, vol. 433, no. 4, **(2005)** January, pp. 392-395.
[5] I. Turnu, M. Marchesi and R. Tonelli, "Entropy of the distribution and object-oriented Software Quality", In Proceedings of the 2012 ICSE Workshop on Emerging Trends in Software Metrics WETSoM '12, **(2012)**, pp. 77-82.
[6] L. Subelj and M. Bajec, "Community structure of complex software systems: Analysis and Applications", Physica A Statistical Mechanics and its Applications, vol. 390, **(2011)** August, pp. 2968-2975.

 

[7] D. Li, Y. Han and J. Hu, "Complex network thinking in software engineering", In Proceedings of the 2008 International Conference on Computer Science and Software Engineering, Washington, DC, USA, 2008. IEEE Computer Society, vol. 01, CSSE '08, pp. 264-268.
[8] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks", Phys Rev. E, **(2004)** February.
[9] G. Concas, M. Marchesi, G. Destefanis and R. Tonelli, "An empirical study of software Metrics for assessing the phases of anagile project", International Journal Soft. Eng. Knowl. Eng, vol. 22, **(2012)**, pp. 525.
[10] C. R. Myers, "Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs", Phys. Rev. E, vol. 68, no. 4, **(2003)** October, pp. 046116.
[11] J. Dietrich, V. Yakovlev, C. McCartin, G. Jenson and M. Duchrow, "Cluster analysis of java dependency graphs", In Proceedings of the 4th ACM symposium on Software visualization, SoftVis '08, New York, NY, USA, ACM, **(2008)**, pp. 91-94.
[12] L. Subelj and M. Bajec, "Community structure of complex software systems: Analysis and applications", Physica A Statistical Mechanics and its Applications, vol. 390, **(2011)** August, pp. 2968-2975.
[13] G. Concas, M. Marchesi, A. Murgia and R. Tonelli, "An empirical study of social networks metrics in object-oriented software", Adv. Soft. Eng., **(2010)** January, pp. 4:1-4:21.
[14] 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, vol. 181, **(2011)**, pp. 883-902.
[15] G. Concas, M. Marchesi, A. Murgia, R. Tonelli and I. Turnu, "On the distribution of bugs in the eclipse system", IEEE Transactions on Software Engineering, vol. 37, no. 6, **(2011)** November, pp. 872-877.
[16] 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, **(2012)**, pp. 63-72.
[17] V. S. C. R. and V. Sole, "Scale free networks from optimal design", Europhysics Letters, vol. 60, **(2002)**.
[18] S. Valverde and R. V. Sole, "Hierarchical small worlds in software architecture arXiv:cond-mat/0307278v2, **(2003)**.
[19] A. L. D. Cahllet, "Bug propagation and debugging in asymmetric software structures", Phys Rev E, vol. 70, **(2004)**.
[20] W. Pan, B. Li, Y. Ma and J. Liu, "Multi-granularity evolution analysis of software using complex network theory", Journal of Syst. Sci. Complex., vol. 24, no. 6, **(2011)**, pp. 1068-1082.
[21] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks", Phys. Rev. E., vol. 69, no. 2, pp. 026113.
[22] L. Wen, D. Kirk and R. G. Dromey, "Software systems as complex networks", Proceedings of the 6th IEEE International Conference on Cognitive Informatics, COGINF '07, Washington, DC, USA, IEEE Computer Society, **(2007)**, pp. 106-115.
[23] M. Hamdaqa, L. Tahvildari, N. LaChapelle and B. Campbell, "Cultural Scene Detection Using Reverse Louvain Optimization", Science of Computer, **(2014)**.
[24] S. Fortunato, "Community detection in graphs", Physics Report, vol. 486, **(2010)** February, pp. 75-174.
[25] A. Clauset, M. E. J. Newman and C. Moore, "Finding community structure in very large networks", Phys. Rev. E, vol. 70, no. 6, **(2004)** December, pp. 066111.
[26] M. E. J. Newman, "The structure and function of complex networks", SIAM REVIEW, vol. 45, **(2003)**, pp. 167-256.
[27] https://en.wikipedia.org/wiki/Modularity_(networks).
[28] B. Shopov (September 8, 2014). "Implement Client-side Bug Reporting". Archived from the original on 13 November 2014. Retrieved 17 November (2014).
[29] http://www.openoffice.org/marketing/art/about-issuezilla.html. In Cvs. http://www.nongnu.org/cvs/.
[30] 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– conference of the center for advanced studies on Collaborative research, CASCON'07, pages215–228, New York, NY, USA, ACM, **(2007)**.
[31] M. Eaddy, T. Zimmermann, K. Sherwood, V. Garg and G. Murphy, "Do Crosscutting concerns cause defects?", IEEE Transactions on Software Engineering, vol. 34, no. 4, **(2008)** November, pp. 497-51.
[32] R. S. I. Gyimothy and T. Ferenc, "Empirical validation of object-oriented metrics on open Source software for fault prediction", IEEE Transactions on Software Engineering, vol. 31, **(2005)**.
[33] M. E. Newman, "Fast algorithm for detecting community structure in networks", Phys. Rev E, vol. 69, no. 6, **(2004)** June, pp. 066133.
[34] R. Albert and A.-L. Baraba´si, "Statistical mechanics of complex networks", Reviews of Modern Physics, vol. 74, **(2002)** January, pp. 47-97.

 