ISSN 2278-3091
Volume 10, No.2, March - April 2021
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse1051022021.pdf
https://doi.org/10.30534/ijatcse/2021/1051022021

Analyzing Implementation Architecture through measures of Social Network Analysis: A case study

Sanjay Bapu Thakare[1]
[1]Dr. Babasaheb Ambedkar Technological University, Lonere, Maharashtra, India, mail2sbt@gmail.com

ABSTRACT

We present a case study of an interpretation and application of the measures in the Social Network Analysis (SNA) to comprehend the implementation architecture of a software system. The architectural structure derived from the dependencies of programming elements is considered a social network. Then, the SNA measures such as degree distribution, centrality, clustering, community, and ego network are applied to the software network. The Gephi tool has been used to collect the values of those SNA measures and also for visualization. The values obtained for those measures were interpreted in the terms of potential application area of the software engineering. The interpretation of SNA measures from the case study reveals that it is useful for the developers to enhance their comprehension about the organization of program elements and source.

Key words: Measures, Social Network Analysis, Software Network, Software Architecture.

1. INTRODUCTION

Building a software system is complex process and requires considerable efforts, high level planning and design. With this intense process produces a software system, which is a solution to the real-world computation problem achieved through decomposition and abstraction techniques. Such system is often decomposed into collection of communicating components or elements to manage its complexities. These components will be implemented using different form of abstractions to wrap details and will be gelled together as a system to perform desired computation. Desired computation is achieved through communication or interactions between the various program elements in the system. Such interactions will connect program elements and form structure of software.

These interactions define a software dependency network, give rise alternative way to understand underlying structure of a software. Researchers investigated such network by considering it as a social network to examine the network satisfies certain properties like degree distribution (scale free), small world, centrality, clustering, and ego network. The network is extracted from different type of software's at various levels of interaction granularity mainly from product (source code) and design artifacts (class diagram) [1]. At the same time, different authors worked on the network captured from design and source code of the software at various granularities such as package or module level [2], component [3], class [4], and method [5]. Interactions among the elements are captured at different phases - at runtime [6] and non-runtime (design level or source code). But software system analyzed till now is either procedural (written in C) or Object-oriented systems (C++ and Java based) which are free and open source [3], involve study of various properties of a network along with evolution and stability of software [7], [8].

Further analysis focused on, ranking and search engine of Java classes [9], measures complexity of network [10], [11] and community structure [12], [13], [14]. Along with this [15], [16] and [17] were suggested implication to software but lacks concrete application and use. However, not only software (interacting elements) but also process to build software can be represented as complex network [16] which includes representation of dependency or communication among developers as network.

Similarly, many natural systems were represented as complex network and analysis of such networks with help of social network theory revealed that poses scale free and small world attribute and categorized in social, information, technological and biological network [18]. First, social networks were examined like collaboration [19], co-authorship [18] and virtual and real social network. Second, common information networks were used in the research such as World Wide Web [20], and citation network [18]. In technological networks like Internet [21], [22], software network [6], [2], [7], [10], [16], [3], [9], [23] and lastly metabolic network such as protein-protein interactions, neural network in biological network [24] study exhibits these properties.

The scale free (power law) refers distribution of degree in the network skewed in nature. The free of scale means the distribution unchanged even scaling independent variable [18],[25]. The skewed degree distribution defines resilience (robustness) property of the network where a network is not disconnected even if a node is removed. Due to the probability of such a node to a peripheral node is more than the hub. Further robustness of the network against any random failure or removal, based on the connectivity of a network, removal of a random node will increase distance in the network [19], [18].

The network also exhibits a small-world property, and it is based on average shortest path as well as clustering coefficient of the node in a network. The network with the small-world property must have large clustering coefficient and small average shortest path [26].

In this paper, we examined various properties of software dependency network at a node and network level such as degree distribution, shortest path, clustering coefficient, centrality, ego and community. Most of these properties are examined by considering additional information provided by the network that is direction but, in some cases, we ignore direction. For this, we construct dependency network of import relation extracted from a source code of object-oriented software. In import relation, dependency between a class and explicitly imported classes is captured. Here, interface is also considered as class. This paper addresses following questions to investigate a complex network extracted from software.

"What is the current structure of a software?"
"Do software networks have properties of a social network?"
"Interpretation of the measures in SNA from perspectives of Software Engineering"

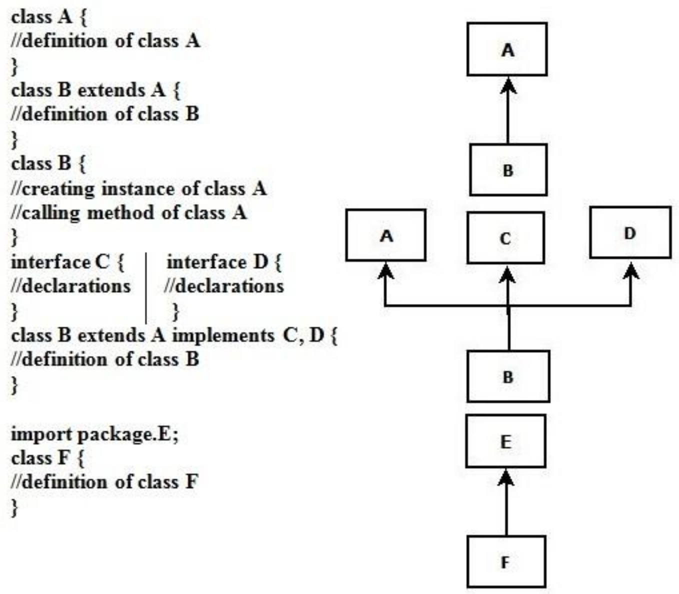
The main contribution and difference between our work and other authors approach are that we have not only apply various measures of SNA to software network (which is done by other authors) but also used these measures to interpret from the perspective of a software.

The remainder section is outline as detailed on approach used to analyzed dependency network of the import relation in object-oriented software, experimental results, interpretation, and related work. Section 2 presents brief of measures in SNA. The approach used for an experiment analysis is described in section 3. Analysis of the result is discussed in section 4. Section 5 presents interpretation of analysis from perspective of software organization and layered program architecture. Section 6 is review of related work on analysis of the social network that has property of scale free and small world. Conclusion is presented in section 7.

2. MEASURES OF SOCIAL NETWORK ANALYSIS

Social network analysis (SNA) is systematic analysis of the pattern and regularities of the relationship in the social network with the help of graph theory, which consists of nodes and ties that connect nodes. An analysis of social network helps to extend our understanding about the relationship and pattern of the relationship that present in the network which cannot be visualize with other techniques.

The pattern of relation is influenced by personality, education, background, race and ethnicity and has significant impact on the individual in various contexts such as social, business, education and politics. Many real-world networks from Web, Internet, collaboration, protein, metabolic, movie actor, cellular, citation, linguistic and power grid have property of scale free and small world. Similarly, software consists of various components that interact and collaborate with each other defines the network. Now researchers are taking interest in to apply concepts of social network to software system in order to understand underlying structure of the software. Such structure of software system pays the important role to support the functionality of software.



```
class A {
//definition of class A
}
class B extends A {
//definition of class B
}
class B {
//creating instance of class A
//calling method of class A
}
interface C {    |  interface D {
//declarations   |  //declarations
}                |  }
class B extends A implements C, D {
//definition of class B
}

import package.E;
class F {
//definition of class F
}
```

Figure 1: Collaboration network of program elements.

The network extracted from design or source code can be either call graph or collaboration (as shown in figure 1) network that helps us to know the inner structure of software. The study of topological properties of large and complex software network used to improve an efficiency and performance of software by guiding a process of testing, debugging and maintenance based on these properties.

The remainder section provides brief about measures in Social Network Analysis.

1) *Degree distribution*: The number of edges connect to a node is degree but degree of all nodes in the network is not apparently same. Thus, degree distribution is described by a function that measure spread of degree in a network. Further, distribution function not only gives the spread of degree in the network but also defines a characteristic of a network that distinguishes it from other networks. At the same time, distribution is probability distribution and defined as ratio of number of nodes with k degree to all the nodes in a network.

$$P(k) \sim k^{-\gamma}$$

However, for a directed network has in-degree and outdegree distribution. But we must conceive in-degree and out-degree in right manner for the network. For example, a call relationship in which a person calls another person, high in-degree indicates that the person receives more calls and high out-degree shows that more call. There are various implications of degree distribution of a network such as if we want to propagate information in the network then obvious choice to leak this information is to a high out-degree node, because it is responsible for spreading information to large number of nodes connected to it. The power law or skewed degree distribution of the real-world network in which portion of large degree nodes are much smaller than low degree nodes [27], [5].

2) *Centrality*: The centrality is a node level measure and used to measures importance of or central nodes in the network. A central node is most influential, and it has great potential to communication and rich access to the information.

Degree centrality: The most influential or popular node is a node with higher degree in the network. It is simplest centrality measure and considers only direct contact between the nodes and ignores indirect contact. Beside simplicity, it considers only immediate connections and overlooks indirect contacts. In directed graph, there are two separate degree centrality measure, in-degree and out-degree based on incoming and out-going edges respectively. Degree centrality of a node is degree of a node "a" and normalized with respect to maximum degree of the node (present in star network*)*.

$$C_D(a) = deg(a)$$

$$NC_D = \frac{C_D(a)}{n-1} = \frac{deg(a)}{n-1}$$

Closeness centrality: To identify a popular node which spread more information in a network quickly; so, distance between the nodes should be small, hence information may reach faster from one node to other. Further, it is defined as an average distance from a node to all the connected node at different depth. So, average distance is measure for the closeness centrality. However, distance between disconnected components of the network is infinite and hence not applied to the disconnected component of a network. For a central node, an average distance is small and calculated as inverse of the sum of distance to all other nodes and normalized closeness (NCC) is in the range from zero to one, zero is interpreted as isolated node and one for strongly connected.

$$C_C(a) = \sum(\frac{1}{d_{ab}})$$

$$NC_C = \sum(\frac{n-1}{d_{ab}})$$

Betweenness centrality: The node connects two or more components, hence acting as bridge or channel to pass the information [28], [29] and gives one more measure to determine a central node, removal of such nodes disconnect the network. Betweenness centrality is define as a number of the shortest paths pass through the nodes.

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is total number of shortest paths from a node s to t and $\sigma_{st}(v)$ is number of paths that pass-through v. The relative betweenness centrality of any node in a graph with respect to the maximum centrality of a node.

$$C'_B(v) = \frac{2C_B(v)}{n^2 - 3n + 2}$$

3) *Average shortest path*: An average shortest path length is small means fast propagation of information and reduces transfer costs and is one of the characteristics of small world network. The distance d_{ij} is distance between any pair of nodes *i* and *j*. *L* is mean distance between any pair of nodes *i* and *j* and define as sum of shortest distance between all pair of node divide by $\frac{1}{2}N(N-1)$[8].

$$L = \frac{2}{N(N-1)} \sum_{i \geq j} d_{ij}$$

4) *Clustering coefficient*: A clustering coefficient (CC) is a measure of degree to which nodes in a graph tend to cluster together. Nodes clustering coefficient is density of its neighborhood and it is measured number of actual edges between neighbors divide by all possible edges. Clustering coefficient of a network is an average of clustering coefficient of all the nodes in a network.

$$CC = \frac{number\, of\, actual\, edges\, in\, neighbors}{all\, possible\, edges}$$

$$\bar{C} = \frac{1}{n} \sum_{i=1}^{n} CC_i$$

Newman [30] in 2001 proposed an alternative calculation for clustering coefficient is known as global clustering coefficient and is defined as number of closed triplets over the total numbers of triplets. Triplet is three connected nodes and there are two kind of triplets. First, open triplet, in which ends are not connected with each other. Second, closed triplet with their ends are connected with each other.

5) *Network Density*: Network density measures cohesion or clustering of a network and defined as ratio of the actual edges to all the possible edges.

$$Network\,Density = \frac{Actual\,Connections}{Possible\,Connections}$$

6) *Ego Network*: An ego network of a node is consisting of all its neighboring nodes and edges connecting them with the aim to study the behavior and role of individual in the network. So, an ego is focal (ego) node and network has ego's equal to total number of nodes present in the network. In directed network, there are in and out kinds of ego network based on incoming and outgoing edges from all the neighbors. In this paper, we consider a network as whole and an ego focuses on the part of network.

7) *Community*: A set of tightly connected nodes are referred as a community like our real-world community or group on social media. Here, study focuses on the sub-structure level where a role and behavior in the group are analyzed. It is fundamentally a partition technique in which network is dividing into groups based on the density of edges within the group and between the groups.

3. COLLECTING SNA MEASURES

This section describes a tool used to extract the dependency relationship from Object-oriented software and finding values of various measures of SNA. A tool was developed in the Java to recognize the import dependency amongst the program elements. The tool implements an algorithm 1 to extract import dependency relationship and store in a suitable format for further analysis. Input to an algorithm is directory path of source code and produces output in different format (gexf, gml or edgelist) compatible to various SNA tools. The step 2 and 3 in the algorithm lists all existing files and sub-directories in input directory of source code. Additionally, checking of all files are carried out and only required files are considered for processing. The process of identifying import relationship from source director is recursive, at each time one source file is opened and retrieve necessary information.

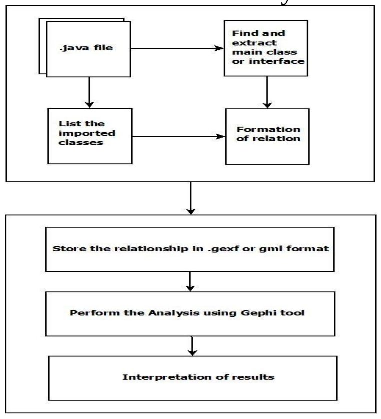

.java file
Find and extract main class or interface
List the imported classes
Formation of relation
Store the relationship in .gexf or gml format
Perform the Analysis using Gephi tool
Interpretation of results

Figure 2: Block diagram of experimental procedure.

The selected file is processed in step 8 to 16, by checking word "import" and "class" in a line read from a file and extracting name of classes from that line. The class name retrieved from import statement recorded in a list. Similarly, a public class in a source file is also retrieved by ignoring an extends and implements the keyword. Further dependency relationship is established from public class name encounter in the file to list of class in import statement. Now, extracted entities are referred as names of nodes in the network and unique identification is assigned to them. A relationship in form of set edges is created between the public class and list of imported classes. Finally, these relationships are stored in the formats necessary for further processing.

Algorithm 1: Extraction of import relation from source code
Input: path of directory contains source code of software.
Output: extracted relations in the format: gexf, gml or edgelist

```
1: procedure EXTRACT–RELATION
2: Open the directory contains source code
3: Listfiles := list all the files in the directory
4:         for each File ∈ Listfiles do
5:             If File is directory then goto 3
6:             else
7:             if File has .java extension
8:               for each Line ∈ File do
9:                   Line := nextline(File)
10:                  If Line start with "import" keyword then
11:                      ImportedClassList := extractclass( Line )
12:                  If Line contains "class" keyword then
13:                      Class := extractclass( Line )
14:                      Classid := checkclass( class ) and goto 17
15:             end for
16:       end for
    Formation of relation:
17:        for each ImportClass ∈ ImportedClassList do
18:            Importclassid := checkclass( ImportClass )
19:            Form edge ( Classid , Importeclassid )
20:       end for
21: end procedure
```

Table 1: Summary of statistical analysis of the ConStore library

Parameters	**With user-def classes**	**With Java classes**
# nodes	66	95
# edges	141	317
Avg. degree	2.136	3.337
Highest in-degree	11	27
Highest out-degree	20	27
Diameter	5	5
Avg. path length	1.584	1.566
Number of shortest paths	245	544
Density	3.3 %	3.5 %
Highest betweenness	35.833	51.8
Highest closeness	2.488	2.5
Avg. clustering coefficient	0.094	0.103

In addition to this, tool support variety of output formats (*node list*, *edge list*, *gexf*, *gml*) attuned with various Social Network Analytical tools. Social Network Analysis tool Gephi was used for getting values of various measures in SNA and recorded in proper format. The output of these tools was recorded for further analysis. The analysis is interpreted from the perspective of software organization and layered program elements. Figure 2 shows the block diagram of the experimental procedure carried out.

4. CASE STUDY: CONSTORE ANALYSIS

ConStore is open-source library that presents storage facility for concept network which model a real-world problem. It provides features like query, store and model the real-world network. The real-world networks are accessed and stored in the form of network where nodes represent the concepts, and edges represent the connection between the concepts.

We selected ConStore library for this case study to understand its internal structure and to verify the properties of the social network. Internal structure of this library is not prior available, so provides an opportunity to uncover the implementation structure of program elements. For the analysis, we focused only on import dependency relationship between the components of a program. The dependency network of import relation is constructed using Algorithm 1 and visualized with assistance of Gephi [36]. During the analysis of dependency network, we have considered the import relationship between: 1) the only user defined classes and 2) including Java built-in (library) classes. Table 1 gives a summary of statistical analysis of the ConStore library. There are 66 user-defined classes and 29 libraries or built-in Java classes, so total 95 classes are in the analysis. We have found 141 import relations between the user defined classes and 317 whereas with built-in classes.

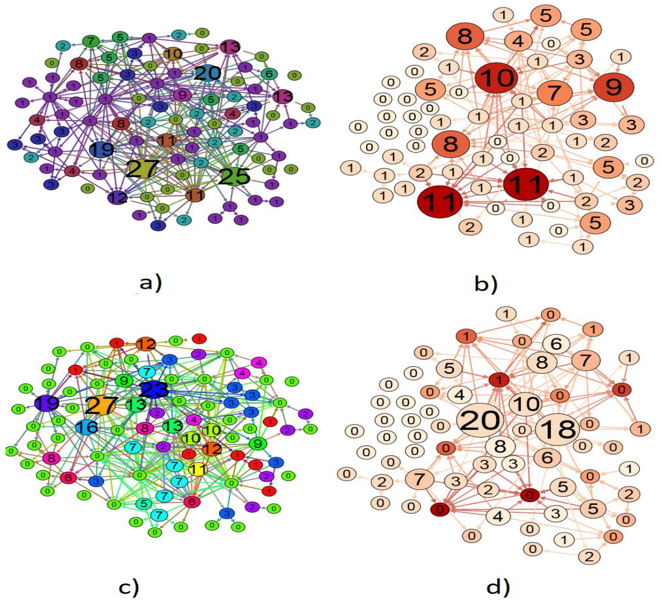


Figure 3: Degree centrality analysis of the ConStore where labels are node id and size correspond to degree.

Due to this a little increase in the density of the network from 3.3% to 3.5%. The diameter of each case is the same that is five which indicated the length of dependency. The number of nodes involved is six. The average shortest path between any two pair of nodes is 1.5, indicates the distance between two classes. The average degree is around three specifies number of connections fall on each node.

Figure 3a and 3b show in-degree of user-defined and with built-in class whereas figure 3c and 3d show out-degree analysis of various classes. Further, we observed that 35(53%) nodes have degree less than equal to three in the network and only two (3%) degree above 17.

The figure 4 reveals presence of skewed degree distribution of the ConStore which clearly shows for both in, and out-degree follows the power law. This means the heavy tail on the left-side of the plot and flat on right-side or head part of the plot.

An ego network FileConceptNet(19) and IOException(node id 22) are shown in figure 5 and 6. The first figure 5a and 6a illustrate the first order ego of the nodes as mentioned above. The second figure 5b and 6a shows the second order private network of the nodes.

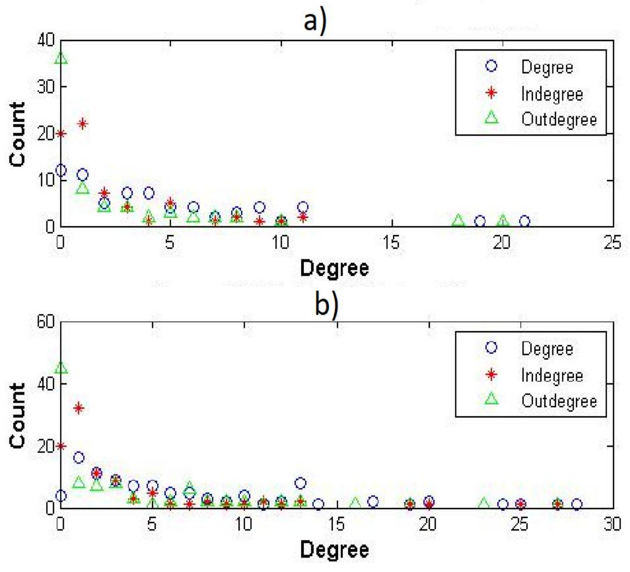


Figure 4: Degree distribution of the ConStore.

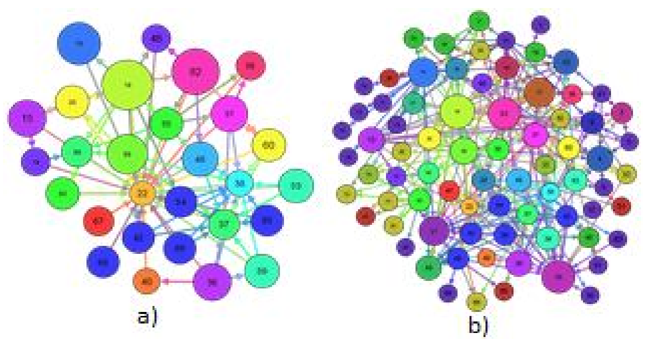


Figure 5: An ego network of *FileConceptNet*(19) with order a) first and b) second.

It is evident from statistical analysis that value of betweenness and closeness properties for Java library classes is zero. This indicates that these nodes are locating at the boundary of a network (peripheral nodes). *BufferedFileAdapter* (11), *Type* (10) and *IOAdapter* (11) are most imported classes while mostly imported classes with considering built-in classes are *IOException* (27), *ByteBuffer* (25) and *List* (20).

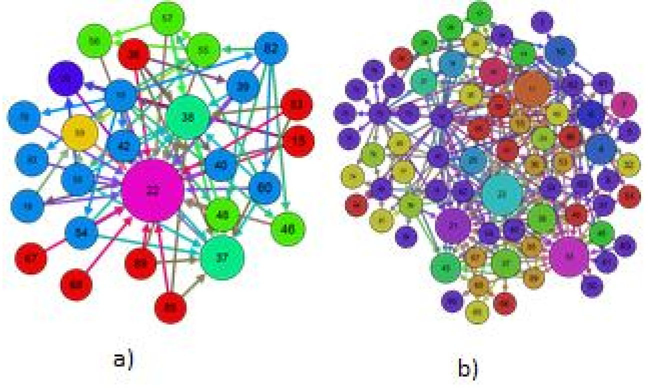


Figure 6: An ego network of IOException (22) with order a) first and b) second

The figure 7 shows the clustering coefficient of the network which reveals the density of neighborhood. The average clustering coefficient (neighborhood edge density) of the network is low.

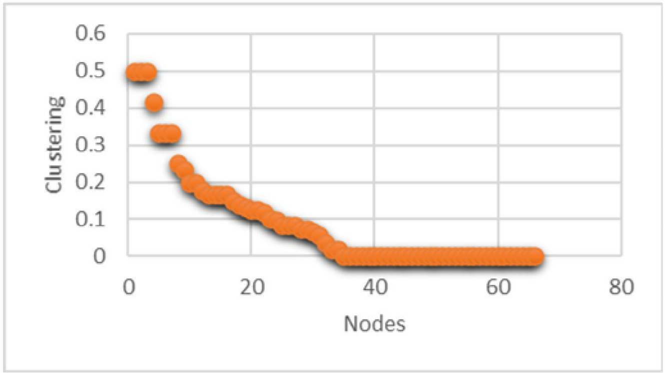


Figure 7: Clustering coefficient analysis of the ConStore.

There are three major communities or groups found in the ConStore library as shown in figure 8. The distribution of nodes in communities is: first large community is 30%, second community is 18% and third large community consist of 8%.

To address the first question, we have recovered the structure of classes in the ConStore library. The figure 9 shows the implementation structure of the ConStore library. We found 13 nodes(classes) did not have any relations with other nodes. These classes are not explicitly imported by the other classes, may exist on implicit relations. In future, we not only consider import relation but tapping all the relations between the classes, so program structure become more accurate.

5. INTERPRETING SNA MEASURES

Degree Distribution: ConStore network has skewed degree distribution (power law) and indicates the system is a fault tolerance(resilience), due to large low degree nodes (peripheral) and fewer high degree nodes(hub). So, the degree distribution of software predicts reliability of software against any random failure. The major functionality was controlled by hub nodes has lower probability of failure. While peripheral nodes have high probability of failure indicating possible less damage to the main functionalities. Thus, resilience property ensures reliability of functionality in the software and that leads to the customer satisfaction.

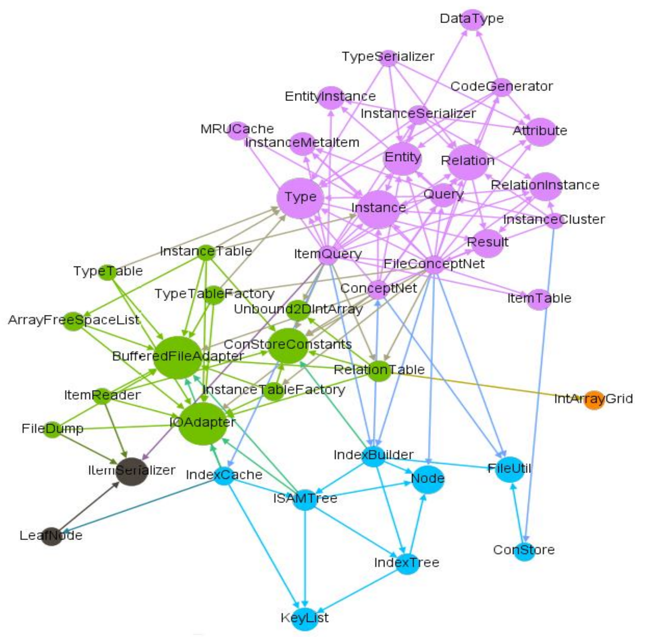


Figure 8: Community analysis of the ConStore.

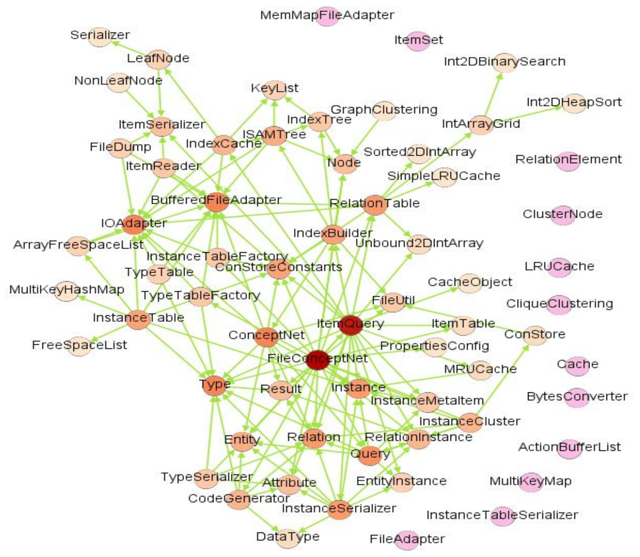


Figure 9: Implementation Architecture of the ConStore.

In-degree: In-degree of class means incoming connection, simply indicates the reusability of the computing services offered by the class. Now, a class(node) with high in-degree mean it has been imported by most of the classes, provide core computing, can be present mostly in utility or library. In case of *ConStore*, mostly reused user-defined classes are *BufferedFileAdapter*, *IOAdapter* and whereas in case of built-in classes are *IOException*, *ByteBuffer*, *ArrayList*, *File List*, *FileNotFoundException*, and Map. All these are part of Java library classes. Thus, identification of highly reusable classes is important for testing and maintenance process because changes made will affect to the classes which depend on it.

Out-degree: A class with high out-degree is an aggregator and a controller of functionality provided by the dependee classes of software and so on greater extent it relays on dependee classes. Therefore, higher priority is given to such class in testing and debugging process as more likely faults presents in these classes. Therefore, these classes are included in 20% of 80-20 rule and responsible for defect clustering. In the maintenance process, these classes are very sensitive to the changes.

Betweenness centrality: A node is more frequently along a path that connects two nodes and has high betweenness value. Thus, a class with high betweenness value is nontrivial and act as bridge to connect parts of system. These classes integrate the system. At the same time, it has been found from analysis of ConStore that most of the classes with high out-degree have high betweenness value. It suggests for refactoring to simplify the class.

Closeness centrality: The closeness centrality is based on the shortest path between the classes. If distance between the classes is small then better the communication and faster the computations. Large distance simply indicates the delay in the communication.

Ego network: An ego network refers to a private network. In a dependency network, an ego of class is useful for analyzing a ripple effect of changes or modifications. However, an ego is also coarse level analysis which has many applications in software engineering such as network partition, structural analysis at node level, and to estimate the properties of a network.

Community: A community is a closely connected group which often contains classes not similar to the classes in the packages. The community allows you to partition the system into modules different from the partition of the packages. As there are multiple ways to divide the system to satisfy the need of customer with different view point. Community provides the new perspective to partition the system and useful for predicting refactoring.

Shortest path: The path length allows us to understand the level of the dependency between the classes. Further, if this because of the inheritance then we must careful that path length should not be large. Because it introduces the communication delay as number of classes in the path increases. We can also assert that shortest path between the classes means computation becomes fast.

Density: The density of the ConStore is less than 4%, in the real-world network it is up to 10%. More number of edges is good for a real-world network but for software each edge is a computation demand. So, it is better that software networks are sparse in nature.

An important implication of these finding is that measures of Social Network Analysis is useful to pull out program architecture. At the same time, it can be also helpful for significantly improve testing, debugging or maintenance processes.

6. RELATED WORK

Network has been investigated more than two decades by research community to deepen understanding of structure used for different purposes [20]. The only intention of many authors to examine a network constructed from different software was to validate whether a network has certain properties of the Social Network. Thus, most common properties that the researcher found to be present in software network were power law degree distribution, small world, scale free properties, and community [2], [3], [4], [5], [6], [7], [8].

Initially, Faloutsos [21] and Alberto Median [22] revealed that Internet followed the power law, despite its apparent randomness and fits well for three snapshots of the Internet taken for experiment. Although Albert Barabas[27], Newman[30] and Reuven Cohen[31] also studied various networks and topologies that exhibit scale free property comparing with random network model.

On the other hand, David Hyland-Wood [5] and Alex Potanin [32] were verified scale free property in network constructed from Java based software. Further, Valverde and Cancho examined an undirected [17] and directed [4] class collaboration network based on the class diagram of JDK 1.2 with 9257 nodes, 3115 connected components and first largest component exhibits scale free and small world.

Now, researches considered various software network constructed from open-source software system written in different languages for analysis [16], [15]. Most commonly studied software systems with help of the social network were Linux, Mozilla, XFree86, Gimps and JDK [4] [17] [34]. However, software network captured various relationships like calling, dependency, common header files, and collaboration present between various programming elements.

Researchers have also studied software network at different level of granularity, starting from inter-package(module) to function level captured at design, compile and run time [2]. In addition to this, a large collection of on-line Java classes available on the Internet ware analyzed by Diego and Fabrizio [9] from eighteen software with intention of finding top ranking classes and components. A class-ranking algorithm similar to Google's Page Ranking algorithm was implemented to rank the classes with aim to efficient search highly relevant classes or components from repository to reduced component search time so as to seed-up component reuse process.

Moreover, software network also exhibits community structure (group of related classes or packages) and was not similar to package organization [33]. Now, application of the community detection technique and when combined with various software metrics reveals various defects in the software system [35]. It helps to assess and improve the system development. Li used the concept of social network to study internal structure in order to supervising various aspects of complex software system [37], [38]. There are various aspects of software system where researchers used social network analysis such as a measuring stability [39], measuring modularity and decomposition [40],[41], class cohesion [42].

At same time several authors shown their interest in examine various properties of software network constructed mostly from open-source software at different level of granularity, phase and relationship. However, very little discuss was carried out on implication of such extensive analysis. Due to this, aim of this paper to discuss implication of various measures in Social Network to software from different perspective like detecting implementation structure, improve organization of components, testing and maintenance process.

7. CONCLUSION

Concepts and measures of the Social Network Analysis have been applied to software system in order to understand underlying structure of software because it pays the important role to support functionality of software. Which is often influence, help to improve and support the maintenance process and evolution of a software system.

This paper clearly shown that study of software network built from the import dependency relationship extracted from ConStore has clearly reveal properties of the social network. The network also represents the program implementation architecture. The finding suggests that measures of SNA help to study the underlying software architecture. Also, it could be applicable to other phase of software development and maintenances process by improve software organization, testing and maintenance process. In future, we not only consider import relation but tapping all the relations between the classes. We try to extend our study to different kinds of software systems and explore more implications of the analysis.

REFERENCES

1. P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, **Graphbased analysis and prediction for software evolution**, in Proceedings of the 34th International Conference on Software Engineering, IEEE Press, 2012, pp. 419–429.
2. N. LaBelle and E. Wallingford, **Inter-package dependency networks in open-source software**, arXiv preprint cs/0411096, 2004.
3. R. Nair, G. Nagarjuna, and A. K. Ray, **Finiteasize effects in the dependency networks of free and openasource software**, arXiv preprint arXiv:0901.4904, 2009.
4. S. Valverde and R. V. Sole, **Hierarchical small worlds in software architecture**, arXiv preprint cond-mat / 0307278, 2003.
5. S. K. David Hyland-Wood, David Carrington**, Scale-free nature of java software package, class and method collaboration graphs**, 5th International Symposium on Empirical Software Engineering,September, Rio de Janeiro, Brazil, pp. 21–22, 2005.
6. C. Chambers, I. Pechtchanski, V. Sarkar, M. J. Serrano, and H. Srinivasan, **Dependence analysis for java**, in Languages and Compilers for Parallel Computing. Springer, 2000, pp. 35–52.
7. H. Li, B. Huang, and J. Lu, **Dynamical evolution analysis of the object-oriented software systems**, in Evolutionary Computation of IEEE World Congress on Computational Intelligence, 2008, pp. 3030–3035.
8. L. Wang, Z. Wang, C. Yang, and L. Zhang, **Evolution and stability of linux kernels based on complex networks**, Science China Information Sciences, vol. 55, no. 9, pp. 1972–1982, 2012.
9. D. Puppin and F. Silvestri, **The social network of java classes**, in Proceedings of the 2006 ACM symposium on Applied computing. ACM, 2006, pp. 1409–1413.
10. Y. Ma, K. He, and D. Du, **A qualitative method for measuring the structural complexity of software systems based on complex networks**, in Software Engineering Conference, 2005. APSEC'05. 12th AsiaPacific. IEEE, 2005, pp. 7–pp.
11. Y. Ma, K. He, D. Du, J. Liu, and Y. Yan, **A complexity metrics set for large-scale object-oriented software systems**, in Computer and Information Technology, 2006. CIT'06. The Sixth IEEE International Conference on. IEEE, 2006, pp. 189–189.
12. Y. Ma, K. He, and J. Liu, **Network motifs in object-oriented software systems**, arXiv preprint arXiv:0808.3292, 2008.
13. L. Subelj, S. Zitnik, N. Blagus, and M. Bajec, **Node mixing and group structure of complex software networks**, Advances in Complex Systems, 2014.
14. E. Ferrara and G. Fiumara, **Topological features of online social networks**, arXiv preprint arXiv:1202.0331, 2012.
15. A. P. De Moura, Y.-C. Lai, and A. E. Motter, **Signatures of small-world and scale-free properties in large**

computer programs, Physical review E, vol. 68, no. 1, p. 017102, 2003.

16. C. R. Myers, **Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs**, Physical Review E, vol. 68, no. 4, p. 046116, 2003.
17. S. Valverde, R. F. Cancho, and R. V. Sole, **Scale-free networks from optimal design**, EPL (Europhysics Letters), vol. 60, no. 4, p. 512, 2002.
18. M. E. Newman, “The structure and function of complex networks,” SIAM review, vol. 45, no. 2, pp. 167–256, 2003.
19. M. E. J. Newman, **Random graphs as models of networks**, arXiv preprint cond-mat/0202208, 2002.
20. A.-L. 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, vol. 281, no. 1, pp. 69–77, 2000.
21. M. Faloutsos, P. Faloutsos, and C. Faloutsos, **On power-law relationships of the internet topology**, in ACM SIGCOMM Computer Communication Review, vol. 29, no. 4. ACM, 1999, pp. 251–262.
22. A. Medina, I. Matta, and J. Byers, **On the origin of power laws in internet topologies**, ACM SIGCOMM computer communication review, vol. 30, no. 2, pp. 18–28, 2000.
23. M. Savic, M. Ivanovi´ c, and M. Radovanovi´ c, **Characteristics of class´ collaboration networks in large java software projects**, Information Technology and Control, vol. 40, no. 1, pp. 48–58, 2011.
24. L. d. F. Costa, O. N. Oliveira Jr, G. Travieso, F. A. Rodrigues, P. R. Villas Boas, L. Antiqueira, M. P. Viana, and L. E. Correa Rocha, **Analyzing and modeling real-world phenomena with complex networks: a survey of applications**, Advances in Physics, vol. 60, no. 3, pp. 329– 412, 2011.
25. Q. Liu and A. T. Ihler, **Learning scale free networks by reweighted l1 regularization**, in International Conference on Artificial Intelligence and Statistics, 2011, pp. 40–48.
26. A. Barrat and M. Weigt, **On the properties of small-world network models**, The European Physical Journal B-Condensed Matter and Complex Systems, vol. 13, no. 3, pp. 547–560, 2000.
27. R. Albert and A.-L. Barabasi, **Statistical mechanics of complex networks**, Reviews of modern physics, vol. 74, no. 1, p. 47, 2002.
28. S. P. Borgatti, **Centrality and network flow**, Social networks, vol. 27, no. 1, pp. 55–71, 2005.
29. D. R. White and S. P. Borgatti, **Betweenness centrality measures for directed graphs**, Social Networks, vol. 16, no. 4, pp. 335–346, 1994.
30. M. E. Newman, S. H. Strogatz, and D. J. Watts, **Random graphs with arbitrary degree distributions and their applications**, Physical review E, vol. 64, no. 2, p. 026118, 2001.
31. R. Cohen and S. Havlin, **Scale-free networks are ultrasmall**, Physical review letters, vol. 90, no. 5, p. 058701, 2003.
32. A. Potanin, J. Noble, M. Frean, and R. Biddle, **Scale-free geometry in oo programs**, Communications of the ACM, vol. 48, no. 5, pp. 99–103, 2005.
33. L. Subelj and M. Bajec, **Community structure of complex softwareˇ systems: Analysis and applications**, Physica A: Statistical Mechanics and its Applications, vol. 390, no. 16, pp. 2968–2975, 2011.
34. Ali & Aksoy Tuysuz, **Analysis of function-call graphs of open-source software systems using complex network analysis**. Pamukkale University Journal of Engineering Sciences, 26(2), 2020.
35. Orru, M., Monni, C., Marchesi, M., Concas, G., & Tonelli, R. **Predicting Software Defectiveness through Network Analysis**. In SATToSE (pp. 36–47), 2015.
36. Bastian, Mathieu, **Gephi: an open source software for exploring and manipulating networks**, 2009.
37. Li, Hao, **Modeling Software Systems as Complex Networks: Analysis and Their Applications**. Mathematical Problems in Engineering 2020.
38. Pan, Weifeng, **Applying complex network theory to software structure analysis**. International Journal of Computer and Systems Engineering 5. 12, 2011, 1634–1640.
39. Pan, Weifeng, **Measuring software stability based on complex networks in software**. Cluster Computing 22. 2, 2019, 2589–2598.
40. Pan, Weifeng, **Analyzing the structure of Java software systems by weighted K-core decomposition**, Future Generation Computer Systems 83. 2018: 431–444.
41. Xiang, Yiming, **Measuring software modularity based on software networks**, Entropy 21. 4,2019, 344.
42. Gu, Aihua, **Measuring object-oriented class cohesion based on complex networks**, Arabian Journal for Science and Engineering 42. 8, 2017, 3551–3561.