Studying Multifaceted Collaboration of OSS Developers and its Impact on their Bug Fixing Performance

Amit Kumar
Department of Information Technology
Indian Institute of Information Technology
Allahabad, India
Email: *amitchandramunityagi@gmail.com*

Mahen Gandhi
Area of Computer Science and Engineering
NIIT University
Neemrana, India
Email: *mahenm.gandhi@st.niituniversity.in*

Yugandhar Desai
Area of Computer Science and Engineering
NIIT University
Neemrana, India
Email: *yugandhard.desai@st.niituniversity.in*

Sonali Agarwal
Department of Information Technology
Indian Institute of Information Technology
Allahabad, India
Email: *sonali@iiita.ac.in*

***Abstract*—Developers often collaborate to fix complex bugs, even in open source software systems (OSS) where collaboration largely occurs through discussions in the bug tracker. The implicit Developer Social Networks (DSN) are created as a result of these discussions. Past research has investigated the usefulness of such DSNs in addressing many Software Engineering problems (e.g. Defect Prediction, Evolution of collaboration patterns, etc.). However, the multifaceted nature of DSNs constructed from bug reports data has been ignored in most of the past studies. That is, in most of the past studies, the link among developers exist only if they comment on the same bug report while in reality, the developers may be connected indirectly (e.g. pair of developers are connected even if they comment on two different bug reports which are associated with the same software component). Such unexplored relationships among developers can be used in defining new measures to identify important developers in the OSS system which otherwise is not trivial to do. In this paper, we study this implicit multifaceted nature of collaborations among developers by extending single layer DSN to Multi-layer DSN (MDSN). Our experiments performed on bug data of Eclipse and NetBeans show that structure of DSNs and their evolution at various layers differ significantly and performance of developers in bug fixing process is not only significantly correlated (Pearson correlation coefficient up to 0.74) with their network centrality scores but also vary across various layers of MDSN signifying their usefulness in determining the crucial and important developers in the software systems.**

***Index Terms*—Developer Social Network, Multidimensional Developer Social Network, Multilayered Developer Social Network, Multifaceted Developer Social Network**

I. Introduction

Issue Tracking Systems are not only used to archive bug reports and the related information but also to help developers to collaborate and have a discussion on issues (bugs or features). Developers typically interact by commenting on bug reports. These interactions form an implicit developer social network (DSN).

Due to the readily available data from issue trackers, researchers have started investigating DSNs to solve software development problems. For example, DSNs have been used to study community structures of software developers and their evolution [1], to categorize bug reports [2], and to help in defect prediction [3].

However, most of these studies explore only one type of links among the developers (e.g. In DSN constructed from bug report data, the developers are connected if they have worked together to fix the same bug report) while they are indirectly connected through various other avenues. For instance, developers who have not commented on the same bug report but have commented on two different bug reports found in the same component of a software product, are indirectly connected. In this paper, we consider many such indirect connections among the developers and build the Multi-layered / Multi-faceted Developer Social Network (**MDSN**). In our Multi-Layered DSN, each layer represents a different DSN which shows the links among developers capturing different types of proximity among them. We believe that a holistic view of these different kinds of proximities among the developers and investigation of Multi-faceted Developer Social Network (MDSN) can elucidate more on the nature of developer collaborations on issue tracking systems.

Towards our goal of investigating MDSN, we first attempt to answer the fundamental question if the structure of DSN at various layers vary significantly from each other. Network Structure of DSN has been characterized by many global social network properties in past studies [4] [5]. We also use global social network properties to characterize and investigate DSN of various layers and hence ask the following research question:

RQ1: How significantly the global network properties of DSN vary across the layers of the MDSN?

Past studies have reported that DSN does not remain invariable and evolve. Studying such evolution of DSN is important as it allows us to comprehend how relationships

among developers evolve. MDSN has many layers of DSN and hence it is important to study and compare the evolution of each DSN. This sheds light on the dynamics of DSNs at each layer. In particular, we pose our second research question as follows:

RQ2: How does the evolution of DSNs differ at each layer? Do some DSNs evolve faster than others?

The first two research questions are posed to investigate if the Multifaceted/Multilayered approach of studying DSN adds some value to the understanding of developer communication structure or not.

However, past studies have used DSN to characterize the traits, performance, and importance of the developers [6] [7] [8] [9]. Node centrality measures in DSN and entropy of developers contributions have been used widely to characterize the importance of developers in the collaboration network of developers [10]. In MDSN, the collaboration happens at various levels and hence it will be interesting to see how the importance of nodes in DSN is associated with their bug fixing performance. To measure the importance of the node (developer), we compute the graph entropy-based measures along with various node centrality measures at each layer of DSN. In particular, we ask our third research question as follows:

RQ3: How significantly various metrics measuring the importance of the developers in DSNs correlate with their bug fixing performance? How significantly these correlations differ across different layers of MDSN?

To answer these research questions, we first construct the Multilayered Developer Social Networks from the bug report data of two popular Java IDE projects-Eclipse and NetBeans. Then we use many global network properties (characterizing the properties of the entire network), node importance based measures and measures to characterize the bug-fixing performance of developers to answer our research questions.

II. Related Work

Leveraging archival data to facilitate ongoing software development is a key tenet in software engineering research. One such data that researchers have started using is the implicit social networks that are created because of developer interactions: when they work on the same file or task or communicate regarding an issue or task. Here we sample a subset of research involving DSNs which are related to our work. For example, Canfora et al. [9] mine data from the mailing lists to identify experienced developers who actively interact with newcomers to identify mentors. Bird et al. [1] on the other hand, have used the DSNs from mailing lists to investigate the social status of OSS participants based on the network structure and the relationship between email and commit activities via these networks. Hong et al. [4] have investigated how a DSN evolves and compared it to the evolution in other social networks like Facebook, Twitter, etc. The network structure properties of DSNs have also been studied to identify the structures that correlate with efficiency in the bug fixing process [5].

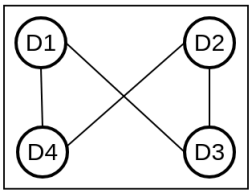

Fig. (1) Single-layered DSN

Zanetti et al. [2] found that centrality of users in a communication network between bug reporters and developers to be indicative of the quality of a bug report. Cataldo and Herbsleb [6] observed that the core developers in the communication structure of the organizations are top contributors. Meneely et al. [3] and Wolf et al. [11] used developer social network for failure prediction. More recently, Wang and Nagappan [12] studied the distribution of collaboration patterns and used them to see the impact of such patterns on the quality of the project from the security point of view.

Our study is similar to the work described above as we also study developer social network. However, instead of using single layer DSN, we study multilayered (Multifaceted DSN). Each layer in our Multifaceted DSN represents different sort of relationship among developers making it a richer framework for depicting more complex proximities among them. Our model of MDSN is inspired by Kazienko et al. [13]. However, to the best of our knowledge, we are first to calibrate and use it in the Software Engineering domain. We also investigate how the positions of developers in networks represented by various layers in proposed Multi-faceted DSN convey about their performance in bug fixing activities. This investigation also adds to the novelty of the proposed work in this paper.

III. Background and Methodology

A. Developer Social Network (DSN) and Multi-layered DSN (MDSN)

Issue Tracking Systems have been used as a communication platform by developers while they work on the bugs reported by users and fellow developers. Developers usually do so by commenting on the bug reports. A typical issue report in ITS of large software ecosystem such as NetBeans or Eclipse has many fields which provides details about the issue e.g. short and long description of the issue, product and component of

TABLE (I) Example Bug Reports

Bug Id	Assigned To	Severity	Priority	Product Id	Component Id	Reporter	Operating System	Commenters
B1	D1	S1	P1	2	3	R1	Linux	D1, D4
B2	D1	S1	P3	1	11	R2	Windows XP	D2, D3
B3	D1	S2	P2	2	3	R3	Linux	D2
B4	D2	S1	P1	2	4	R1	Mac OS X	D2, D4
B5	D3	S3	P3	3	11	R3	Mac OS X	D1, D3

TABLE (II) Dataset

	Duration	#Bug Report's	#Assignees	#Comments	#Commenters	#Reporters	#OS	#Products	#Components	#Versions
Eclipse	2001-2005	27213	759	149340	2767	1911	23	59	325	88
NetBeans	2001-2005	21345	374	134040	1905	1559	16	36	328	21

the given product where the issue is likely to be present, operating system (the product is used/tested with), priority of the issue, severity of the issue, reporter who reported the issue, assignee (whom the issue is assigned to) etc. The discussion on Issue Reporting Systems has been leveraged to construct the Developer Social Network among developers. However, the DSN can be modeled as a single-layered DSN as well as a multilayered DSN. Most of the past studies [4] [5] [8] consider DSN as single-layered, where developers are nodes and edges between the pair of developers exist if they have commented on the same bug report.

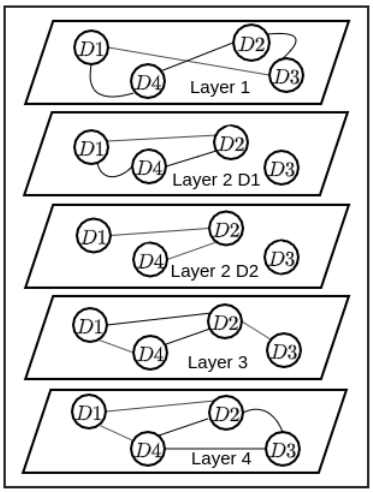

Fig. (2) Multi-layered DSN

We model our DSN as Multi-layered DSN so that multiple facets of the collaboration among developers can be investigated. In our MDSN, each layer represents a different kind of relationship among the developers. In the case of single-layer DSN (DSN considered by past studies), the developers are connected with the edge between them if they have commented on the same bug report. In MDSN, developers are also connected even if they comment on different bugs/issues which are found in the same product, the same component of the product, reported by the same reporter or if the bugs were discovered while software product was used with the same operating system. In total, we have five layers in our MDSN. To illustrate further the difference between single-layered DSN and MDSN, let us consider a toy example data set shown in Table I. There are five bug reports with some (relevant to our study) of their attributes in this table. The single-layered DSN constructed out of this dataset is shown in Figure 1 while MDSN consisting of five layers can be constructed as shown in Figure 2. It can be noted that single-layered DSN is contained in the MDSN as one layer (L1) in it, making it a richer network framework to depict the deeper relationships among the developers. Other layers in MDSN of Figure 2 can be understood easily i.e. L2-D1 represents the network where developers are connected with an edge if they have commented on different bug reports found in the same product. L2-D2, L3, and L4 represent the similar semantics i.e. developers in L2-D2, L3 and L4 are connected with the edge between them if they have commented on the different bug reports found in same product as well as same component, two bug reports reported by same reporter, two bug reports associated with same operating system respectively. It should be noted that we deliberately avoided the trivial links in layers L2-D1, L2-D2, L3, and L4 (replication of links in layers L2-D1, L2-D2, L3, L4 due to L1) by connecting only those developers commenting on different bug reports. We did it to analyze the exclusive nature and power of DSN at each layer. We used DSN at each layer to answer our research questions by exploring the global network properties and various node importance measures of it.

B. Global Network Properties

To investigate the difference in the nature and evolution of various DSNs at every layer, we used similar global network properties as used by Hong et al. [4]. We compared the DSNs at each layer based on the following global network properties:

1) Network density: Network density is defined as the ratio of number of edges present in the network and the maximum possible edges which can exist in the network (excluding self-loops). Higher density of network indicates higher levels of inter-developer communication.

2) Modularity: Modularity of network is important measure as higher value of modularity denotes the higher community structure present in the network. We used the same modularity definition as defined by Newman [14].

$$M = \sum_{i=1}^{n} \left(x_i - y_i^2\right)$$

where, x_i denotes the proportion of the edges between the vertices of the community i while y_i denotes the proportion of the edges that are not part of the community.

3) Average Path Length (APL): It is the average length of shortest paths between each pair of nodes in the network. Shorter APL shows that Developers are well connected to each other in the network and are easily accessible to each other.

4) Average Clustering Coefficient: The Clustering Coefficient in a graph is the degree of clustering by a node with its neighboring nodes. The clustering coefficient of a vertex can be defined as follows in an undirected graph:

$$P_i = \frac{2x_i}{y_i\left(y_i - 1\right)}$$

Here x_i is the number of edges between neighbors of node i and y_i is the number of node i's neighbors. Average

Network Clustering Coefficient is the average of all network nodes clustering coefficients. A significantly higher average clustering coefficient indicates that network follows small world phenomenon [15].

5) Average Degree (AD): The degree of a node in the graph is total number of edges incident on that. Since each edge has two vertices and counts in the degree of both vertices, the average degree of the undirected graph is defined as :

$$AD = 2 \times \frac{|E|}{|V|}$$

C. Node Importance Measures

Past studies have leveraged the position of developers in DSN to investigate their importance in the developer community. In particular, various node based centrality measures in DSN have been used to measure the importance of the developers in DSN. To answer our third research question, we used following node importance measures defined for DSN:

1) Eigenvector Centrality: In graph theory, eigenvector centrality (also called eigencentrality) is a measure of a node's importance in a network. The idea is to assign proportional score values to all network nodes. Let $G(V, E)$ be a graph, consisting of vertices V and edges E. Let A = (a_v, t) be the adjacency matrix, i.e $a_{v,\,t}$ = 1 if vertex v is linked to vertex t, and $a_{v,\,t}$ = 0 otherwise. The relative centrality score of vertex v as defined by Phillip Bonacich [16] is:

$$x_{\mathrm{v}} = \frac{1}{\lambda} \sum_{t \epsilon M(v)} x_{\mathrm{t}} = \frac{1}{\lambda} \sum_{t \epsilon G} a_{\mathrm{v,t}} x_{\mathrm{t}}$$

where $M(v)$ is a set of the neighbours of v and λ is a constant. Mathematically, this can be written in vector notation as the famous eigenvector equation,

$$Ax = \lambda x$$

The principal eigenvector of the above equation denotes the centrality of all network nodes (here, node is the developer).

2) Betweenness Centrality: The Betweenness Centrality of the graph is determined by the propensity of a single vertex to be more central than any other vertex in the graph. In other words, it measures how often a node appears on shortest paths between nodes in the network. The following is the standard measure given by Freeman [17]:

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

where, σ_{st} is the number of shortest paths from $s \epsilon V$ to $t \epsilon V$.

3) Closeness Centrality: A node's closeness centrality in a connected graph is a measure of centrality in a network, measured as the reciprocal sum of the shortest path length between the node and all other nodes in the graph. Therefore, the more central the node, the closer it is to all the other nodes. The closeness centrality as defined by Sabidussi [18] is as follows:

$$C_C(V) = \frac{1}{\sum_{t \epsilon V} d_G(v, t)}$$

4) Entropy based measure: The entropy of a system measures the randomness or uncertainty in it. The entropy of a graph measures the diversity of edges incident on its nodes. Higher the entropy of the graph, more uniform is the distribution of its edges on its nodes. Dehmer and Mowshowitz [19] provides good survey on graph entropy. Graph entropy can be used to measure the importance of the nodes in the graph. For instance, if entropy of a graph is significantly changed after removing a node from it, node is considered important. Though many definitions are available for graph entropy in literature, we define entropy of a graph as follows: Let $G = (V, E)$ be an undirected graph. The entropy of the graph G, denoted as $H(G)$ is defined as:

$$H(G) = \sum_{i \epsilon V} -p_i \log p_i$$

where, p_i is the degree of node i divided by the sum of the degrees of all nodes in the graph. The graph entropy based importance $H(G_i)$ of node i is defined as:

$$H(G_i) = |E_2 - E_1|$$

where, E_1 is the entropy of the graph G with node i and E_2 is the entropy of the graph G without node i. This helps us in determining how important a node is in the graph. Higher the value of $H(G_i)$, more is the importance of the developer i.

D. Performance of developers in bug fixing process

To answer our third research question, we require two set of measures - Node importance measures as defined in previous sub section and measures to quantify the performance of the developers in bug fixing process. We used following measures to measure the performance of developers in bug fixing process.

1) Average fix time: The Average Fixed Time for an assignee **a** is estimated over a certain period of time using the equation below. To calculate the developer's efficiency in certain time period, we only consider the bugs that are opened and fixed during that time period.

$$AFT = \frac{\sum_{i=1}^{n} t2_{\mathrm{b_i}} - t1_{\mathrm{b_i}}}{n}$$

where,

$\mathbf{b_i}$ = $\mathrm{i^{th}}$ Bug in set of bugs assigned to the assignee **a**.

n = Total bugs assigned to the assignee **a**.

t1 = Time when the bug was assigned to the assignee.

t2 = Time when the **FIXED** label was added to the bug report for the first time.

2) Aggregate Priority Points: This metric is used to measure the importance of the developer with respect to the type of bugs he/she fixes. The developer who fixes the bugs with higher priority is considered to be more important. We assign priority points to each developer based on the types of bugs he fixes. First we assign the weightage to each priority type as follows:

TABLE (III) Weightage of Priorities

Priority	P1	P2	P3	P4	P5
Points	5	4	3	2	1

Then we calculate the priority points for each developer. Higher value of priority points for the developer signifies that he fixes the bugs with relatively higher priorities making him more important developer than those with lower priority points.

The equation for estimating the aggregate priority points of developer over a certain period of time is as follows:

$$APP = \frac{\sum_{i=1}^{n} p_{\mathrm{i}} \times c_{\mathrm{p}}}{n}$$

where,

n = Total number of bugs assigned to assignee **a**.

$\mathbf{p_i}$ = Priority of bug.

$\mathbf{c_p}$ = Points allocated to priority p.

3) Aggregate Severity Points: This is very similar to the Aggregate Priority Points. The points allocated for each severity are as follows.

TABLE (IV) Weightage of Severities

Severity	trivial	minor	normal	major	critical	blocker
Points	1	2	3	4	5	6

Note that NetBeans and Eclipse allow users to demand new features that are not technically real bugs. Therefore, we do not consider those bug reports where the severity attribute is set for enhancement because this category is reserved for feature requests or improvements to the product. The formula for calculating the aggregate severity points is as follows:

$$ASP = \frac{\sum_{i=1}^{n} s_{\mathrm{i}} \times c_{\mathrm{s}}}{n}$$

where,

n = Total number of bugs assigned to assignee **a**

$\mathbf{s_i}$ = Severity of bug

$\mathbf{c_s}$ = Points allocated to severity s

4) Total Components Developer Works Upon: This measure is the total number of modules/components that the assignee has worked on during certain time period. This denotes the diversity in the work profile of the developer.

E. Experimental Set up and Dataset

To carry out our work,we performed our experiments with bug reports of two common open-source software projects-Eclipse and NetBeans. We chose these projects because they are very popular among the community of software engineering, developed around a similar time as OSS projects and have similar functionalities that make them a good choice to test our proposed Multi-layered Developer Social Network. In total, Our dataset has 283380 comments made upon 48258 bug reports between 2001 and 2005. We chose this period as both the projects during this time were in their initial phase making them ideal to study their evolution. The complete details of the dataset are shown in Table II.

In Issue Tracking System like Bugzilla (used by Eclipse and NetBeans), though the issue is assigned to one person i.e. Assignee, it is fixed collaboratively by OSS contributors. The collaboration happens through comments made on the bug report. Hence in answering our research questions, we constructed single-layered and Multi-layered DSN out of the comments of the bug reports in our dataset. However, it should be noted that to answer our RQ3, the node importance measures and performance metrics are calculated and analyzed only for assignees as they are most responsible for fixing the assigned bug/issue. We conjecture that the assignees with good node importance measure in MDSN are good performers. In particular, node importance measures of an assignee in different layers influences her performance differently. For computing various global network-based metrics and node importance based measures we used Gephi Network Analysis tool [20].

IV. Results and Discussion

In this section, we discuss our results and their implications with respect to our research questions.

RQ1: *How significantly the global network properties of DSN vary across the layers of the MDSN?*

To answer this research question, we computed all the network measures defined in section III-B with the help of Gephi [20] and compared the global network properties of DSNs formed at each layer of MDSN. Past research [4] [5] has also used the similar approach to compare various networks.The difference in network measures across various layers of MDSN signifies the importance of individual layers in MDSN making it more useful framework to study the collaboration patterns among developers. Our results are shown in Figure 3. It can be seen from the figure that while density, average path length, modularity of the DSN vary significantly across different layers of MDSN, the clustering coefficient remains relatively stable across the layers. The modularity of the network describes the community structure of the network and past studies have leveraged modularity of DSN for community detection and discovering team structures in OSS projects [1] [21] [4]. Variation in modularity across different layers suggest that the different community structure and team structure could be discovered using our MDSN approach improving the knowledge about the team structure in OSS maintenance activities. Furthermore, other network measures e.g. network density, average path length etc. have been used to predict the defects in software modules [11] and hence it would be interesting to investigate if the accuracy of defect prediction models could also be improved by incorporating network measures computed based on MDSN. In nutshell, it is clear from our results shown in Figure 3 that network structure of DSNs formed at various layers is significantly different from each other and encourages to leverage MDSN to investigate their usefulness in solving popular research problems e.g. community detection, defect prediction etc.

RQ2: *How does the evolution of DSNs differ at each layer? Do some DSNs evolve faster than others?*

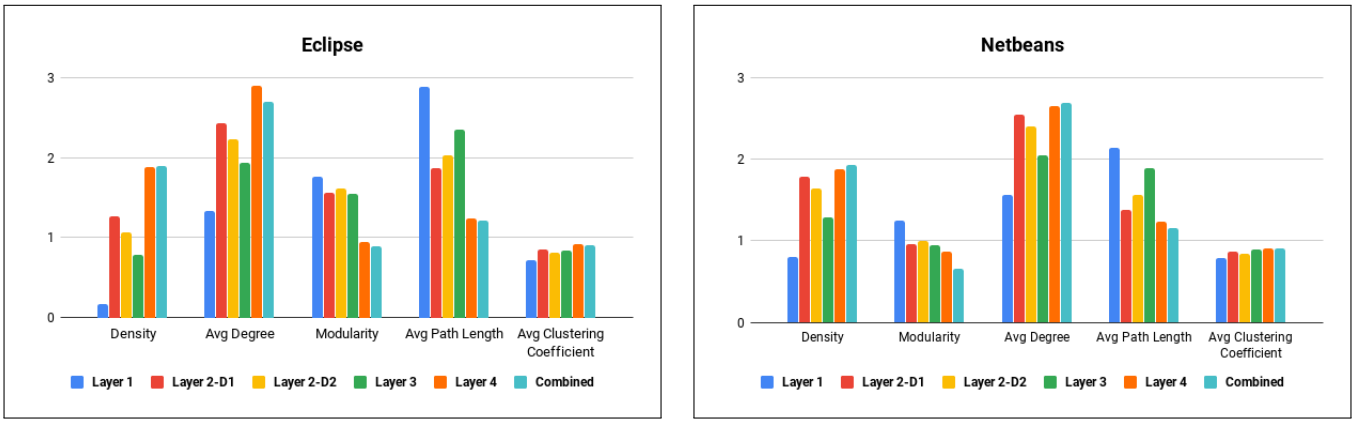

Fig. (3) Variation in Network Structure across various layers of MDSN

To answer this research question, we first split 5 years bug report data of Eclipse and NetBeans (2001-2005) into chunks of 6 months data. This way, we have 10 samples of bug report data for each of the projects. Then we compute the global network properties for each sample to see their evolution over time. Figure 4 shows the evolution of network properties of DSN over time for both Eclipse and NetBeans. It can be seen easily from the figure that the evolution of almost all the network properties at layer-L2-D1 (edges between developer nodes if they comment on different bug reports associated with the same product) and L2-D2 (edges between developer nodes if they comment on different bug reports associated with the same product as well as same component) evolve faster than properties at other layers. This out-performance of these layers is observed for both NetBeans as well as Eclipse. Graph-based metrics such as density, modularity and their evolution have been used to study the software evolution and predicting some of its important aspects [22]. Our study extends the past study on DSN based software evolution and suggests that studying the evolution of DSN at various layers of MDSN can provide new insights to the software evolution research. Variation in evolution of DSNs at various layers also suggests that predicting future aspects of some DSNs is more difficult than others. For instance, predicting the developers leaving the project might be much more difficult in layer L2-D1 and L2-D2 in comparison of other layers where evolution is relatively smoother. Overall, the answer to this research question is affirmative based on the results shown in Figure 4 adding the value to our study.

RQ3: *How significantly various metrics measuring the importance of the developers in DSN correlate with their bug fixing performance? How significantly these correlations differ across different layers of MDSN?*

TABLE (V) Correlation Analysis

Layers	Metric	Avg Fixed Time		Total Components		Aggregate Priority Points		Aggregate Severity Points	
		Eclipse	NetBeans	Eclipse	NetBeans	Eclipse	NetBeans	Eclipse	NetBeans
L1	Betweenness Centrality	-0.079	-0.161	**0.209****	**0.506****	**0.742****	**0.563****	**0.744****	**0.578****
	Closeness Centrality	**-0.195****	**-0.434***	**0.181****	**0.646****	**0.336****	**0.636****	**0.338****	**0.648****
	Eigenvector Centrality	**-0.161***	**-0.443***	**0.315****	**0.656****	**0.482****	**0.656****	**0.475****	**0.651****
	Entropy Based Measure	**-0.162***	-0.222	**0.409****	0.005	**0.635****	-0.087	**0.631****	-0.087
L2 - D1	Betweenness Centrality	-0.059	-0.171	**0.282****	**0.506****	**0.269****	0.358	**0.261****	**0.396***
	Closeness Centrality	**-0.141***	**-0.386***	0.092	**0.646****	**0.162***	**0.472***	**0.157***	**0.489****
	Eigenvector Centrality	-0.096	**-0.424***	0.063	**0.451***	0.032	**0.442***	0.03	**0.448***
	Entropy Based Measure	**0.162***	0.026	-0.097	0.045	**-0.146***	-0.061	**-0.139***	-0.029
L2 - D2	Betweenness Centrality	-0.064	-0.119	**0.297****	**0.48****	**0.275****	0.294	**0.269****	0.326
	Closeness Centrality	**-0.141***	**-0.434***	0.106	**0.521****	**0.175****	**0.464***	**0.172***	**0.473****
	Eigenvector Centrality	-0.092	**-0.384***	0.086	**0.48****	0.035	**0.452***	0.034	**0.485****
	Entropy Based Measure	**-0.156***	0.022	**0.10***	-0.212	**0.191****	-0.093	**0.185****	-0.092
L3	Betweenness Centrality	-0.11	-0.18	**0.223****	**0.468***	**0.74****	**0.642****	**0.745****	**0.647****
	Closeness Centrality	**-0.271****	-0.314	**0.274****	**0.641****	**0.364****	**0.651****	**0.366****	**0.654****
	Eigenvector Centrality	**-0.242****	**-0.523****	**0.311****	**0.553****	**0.357****	**0.526****	**0.356****	**0.525****
	Entropy Based Measure	**-0.232****	-0.212	**0.365****	0.08	**0.474****	-0.01	**0.472****	-0.012
L4	Betweenness Centrality	**-0.25****	**-0.631****	**0.246****	**0.524****	**0.428****	**0.45***	**0.436****	**0.466***
	Closeness Centrality	**-0.347****	**-0.656****	**0.329****	**0.442***	**0.285****	**0.416***	**0.287****	**0.422***
	Eigenvector Centrality	**-0.393****	**-0.788****	**0.279****	**0.385***	**0.204****	**0.371***	**0.203****	**0.373***
	Entropy Based Measure	**-0.39****	-0.365	**0.301****	0.164	**0.251****	0.213	**0.252****	0.222
Combined DSN	Betweenness Centrality	**-0.263****	**-0.656****	**0.255****	**0.438***	**0.32****	**0.402***	**0.325****	**0.407***
	Closeness Centrality	**-0.358****	**-0.711****	**0.277****	**0.397***	**0.242****	**0.379***	**0.243****	**0.38***
	Eigenvector Centrality	**-0.424****	**-0.743****	**0.247****	**0.376***	**0.192****	0.36	**0.192****	0.36
	Entropy Based Measure	**-0.418****	-0.243	**0.26****	0.125	**0.216****	0.115	**0.218****	0.137

$*p < .05$, $**p < 0.01$, $***p < .001$

Fig. (4) Layer-wise evolution of global network properties of DSN (■ Eclipse, ■ NetBeans) from 2001 to 2005.

The main findings of this paper is the answer to this research question. To answer this research question, we first computed various node importance measures for the DSN at each layer as defined in section III-C and the measures to characterize the performance of the developers as defined in section III-D. Then we used the Pearson correlation coefficient to see how effectively and strongly these two sets of measures correlate with each other. We chose Pearson Correlation Coefficient to see the strength of association between two types of measures as it has been found useful by many past studies [23] [24]. Table V shows a summary of our results. The values of correlation coefficients where the p-value is less than 0.05 are specifically highlighted. To see the cumulative effect of node importance measures on the performance of the developers we merged the DSNs of all the layers into one. In this merged integrated DSN, a node exists between the pair of the developers if there exists a link between them in any of the layers - L1, L2-D1, L2-D2, L3, L4. In general, our results are encouraging. For instance, the Eigenvector centrality value of a node in DSN is negatively correlated with the average fix time suggesting that developers who enjoy good eigenvector centrality value fix the issue faster than their peers with lower eigenvector centrality. The value of the correlation coefficient between Average fix time and other node importance measures is also significant. It can also be seen that the correlation coefficient between Average fix time and other node importance measures is maximum for layer-4 out of all the individual layers. This is interesting because past studies show that predicting the fix-time of a bug in OSS is hard. Bhattacharya and Neamtiu [25] reported that many of the features/attributes considered by researchers to build the predictor in predicting the average fix time of the bug are not found relevant. Our results suggest that node importance based measures for assignee can prove to be good features for such predictors. Second, most of the past studies considered the node-based centrality at layer-1 only while our study suggests that similar centrality measures perform better if we leverage MDSN instead of single-layered DSN.

A closer look at Table V shows that node importance measures are also significantly correlated with the total number of components the developer worked upon in fixing the issues. This suggests that developers with high node importance measures gain diverse expertise in fixing the issues making them more crucial/important for the organization. A significant correlation between node importance measures of developers and aggregate priority points as well as aggregate severity points of the bugs fixed by them shows that node importance measures selected in our study are good measures to identify crucial and important developers (developers who are good to fix the bugs with high priority and high severity).

Interestingly, MDSN approach of investigating the impact of various node importance measures on their bug fixing performance provides new insight as many of the results are counter-intuitive, e.g. best correlation is found for layer-4 (where developers are connected if they have commented on

two different bug reports associated with the same operating system). Though the correlation is found significant for both the projects making the finding general enough, the value of correlation coefficients are found to be higher with NetBeans data.

Overall, there are two takeaways from our results-first, significant correlation between various node-based measures and performance of developers suggests that these measures can be used to identify important developers. Second, the measures based on different layers of MDSN are differently correlated with the performance of developers, making the MDSN framework worthy enough to try for identifying the crucial and important developers in OSS.

V. Conclusion and Future Work

In this research, we proposed MDSN to investigate the multifaceted nature of collaboration among the developers while they fix the bugs and collaborate through the Issue Reporting System. There are many takeaways from our research. First, since the structure of networks varies significantly across various layers of MDSN, replicating the past studies on community detection and identifying team formation in OSS on the MDSN framework may provide new insights. Second, Our results show that many node importance measures i.e. node centrality based metrics and graph entropy-based measures have a significant correlation with the performance of the developers in the bug fixing process. Further, such correlations vary significantly across the layers suggesting that MDSN could be more useful to identify important and crucial developers in the developer community of OSS. Though our results are consistent with both the case studies, we selected for our research, there are few threats to its validity. First, comments are not made only by developers on ITS (Issue Tracking Systems) and hence considering all commenters as developers could be a threat to the validity of our results. Second, ITS is not the only platform where developers collaborate. Past research has also used version control data to study collaboration among developers. Hence, performing our study on version control data can complement our study. We plan this in our future work.

References

[1] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 international workshop on Mining software repositories*, pp. 137–143, ACM, 2006.

[2] M. S. Zanetti, I. Scholtes, C. J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: a case study on four open source software communities," in *Proceedings of the 2013 International Conference on Software Engineering*, pp. 1032–1041, IEEE Press, 2013.

[3] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp. 13–23, ACM, 2008.

[4] Q. Hong, S. Kim, S. C. Cheung, and C. Bird, "Understanding a developer social network and its evolution," in *2011 27th IEEE international conference on software maintenance (ICSM)*, pp. 323–332, IEEE, 2011.

[5] A. Kumar and A. Gupta, "Evolution of developer social network and its impact on bug fixing process," in *Proceedings of the 6th India Software Engineering Conference*, pp. 63–72, ACM, 2013.

[6] M. Cataldo and J. D. Herbsleb, "Communication networks in geographically distributed software development," in *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, pp. 579–588, ACM, 2008.

[7] M. Joblin, S. Apel, and W. Mauerer, "Evolutionary trends of developer coordination: A network approach," *Empirical Software Engineering*, vol. 22, no. 4, pp. 2050–2094, 2017.

[8] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 25–35, IEEE, 2012.

[9] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "Who is going to mentor newcomers in open source projects?," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, p. 44, ACM, 2012.

[10] Q. C. Taylor, J. E. Stevenson, D. P. Delorey, and C. D. Knutson, "Author entropy: A metric for characterization of software authorship patterns," in *Third International Workshop on Public Data about Software Development (WoPDaSD08)*, p. 6, 2008.

[11] T. Wolf, A. Schroter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Proceedings of the 31st International Conference on Software Engineering*, pp. 1–11, IEEE Computer Society, 2009.

[12] S. Wang and N. Nagappan, "Characterizing and understanding software developer networks in security development," *arXiv preprint arXiv:1907.12141*, 2019.

[13] P. Kazienko, K. Musial, and T. Kajdanowicz, "Multidimensional social network in the social recommender system," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 41, no. 4, pp. 746–759, 2011.

[14] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, p. 440, 1998.

[16] P. Bonacich, "Power and centrality: A family of measures," *American Journal of Sociology*, vol. 92, no. 5, pp. 1170–1182, 1987.

[17] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977.

[18] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, pp. 581–603, Dec 1966.

[19] M. Dehmer and A. Mowshowitz, "A history of graph entropy measures," *Information Sciences*, vol. 181, no. 1, pp. 57–78, 2011.

[20] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks," in *Third international AAAI conference on weblogs and social media*, 2009.

[21] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp. 24–35, ACM, 2008.

[22] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 419–429, IEEE, 2012.

[23] Z. Zhang, W. K. Chan, T. Tse, P. Hu, and X. Wang, "Is non-parametric hypothesis testing model robust for statistical fault localization?," *Information and Software Technology*, vol. 51, no. 11, pp. 1573–1585, 2009.

[24] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Third International Workshop on Predictor Models in Software Engineering (PROMISE'07: ICSE Workshops 2007)*, pp. 9–9, IEEE, 2007.

[25] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, pp. 207–210, ACM, 2011.