Using Social Network Analysis for Mining Collaboration Data in a Defect Tracking System for Risk and Vulnerability Analysis

Ashish Sureka, Atul Goyal, Ayushi Rastogi
Indraprastha Institute of Information Technology (IIIT)
New Delhi, India
{ashish, atul08015}@iiitd.ac.in, technocrat.ayushi@gmail.com

ABSTRACT
Open source software projects are characterized as self-organizing and dynamic in which volunteers around the world primarily driven by self-motivation (and not necessarily monetary compensation) contribute and collaborate to a software product. In contrast to close source or proprietary software, the organizational structure and task allocation in an open source project setting is unstructured. Software project managers perform risk, threat and vulnerability analysis to gain insights into the organizational structure for de-risking or risk mitigation. For example, it is important for a project manager to have an understanding of critical employees, core team, subject matter experts, sub-groups, leaders and communication bridges.

Software repositories such as defect tracking systems, versioning systems and mailing lists contains a wealth of valuable information that can be mined for solving practically useful software engineering tasks. In this paper, we present a systematic approach to mine defect tracking system for risk, threat and vulnerability analysis in a software project. We derive a collaboration network from a defect tracking system and apply social network analysis techniques to investigate the derived network for the purpose of risk and vulnerability analysis. We perform empirical analysis on bug report data of Mozilla Firefox project and present the results of our analysis. We demonstrate how important information pertaining to risk and vulnerability can be uncovered using network analysis techniques from static record keeping software archive such as the bug tracking system.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous; D.2.9 [Software Engineering]: Management—Productivity, Programming teams

General Terms
Algorithms, Experimentation, Measurement

Keywords
Mining Software Repositories, Social Network Analysis, Collaboration Network, Defect Tracking System, Risk, Threat and Vulnerability Analysis

1. INTRODUCTION
Software development is a knowledge and human intensive task. Teamwork, shared tasks, interaction or collaboration between team members is integral to software engineering. An understanding of aspects like team structure and topology, critical employees, core team, subject matter experts, degree of centralization or de-centralization, sub-groups, leaders and communication bridges is important for an IT (Information Technology) manager or a team leader. For example, employee attrition is common (and inevitable) and a planned approach can save lot of time and energy to an organization in case of employee turnover. Similarly, imagine a situation where an expert is unavailable due to unforeseen circumstance and a manager needs to urgently find a replacement (somebody who is the best fit for resolving the problem at hand) to get the job done. It is important for an organization to know who are the critical employees or leaders and gatekeepers - people who are important and have exclusive knowledge or skills. A good and accurate knowledge of people, their skills, position, role, expertise can be useful to an organization to de-risk itself and enables intelligent and proactive decision making. Assessing risk in a software development environment and taking appropriate action in a timely manner can reduce loss and save time. Tools and techniques to aid a project manager to proactively identify risks and actionable information can be important for project success. However, risk, threat and vulnerability analysis is challenging and an arduous task. In contrast to a commercial organization or an industrial setting developing proprietary software, an open source environment (typically self-organizing, consisting of self-motivated volunteers, absence of face-to-face meetings) poses additional challenges because of the following reasons:

- In general, Open Source Software (OSS) Projects do not follow a pre-designed organizational structure and
is usually dynamic, self-organizing, latent, and usually not explicitly stated [3].

- Constructing an explicit social or socio-technical network representation (relationship and ties between people and between people and technical artifacts) within an organization is a non-trivial task [25]. Imagine a large and complex software product development consisting of several hundred developers working in a fast-paced and a dynamic environment.

- It is non-trivial to understanding the communities that build and support FLOSS software [5].

- Open Source Software (OSS) projects are not formally organized (there is no pre-assigned command and control structure) [3].

- The culture and process in an OSS environment is different than a CSS environment. OSS is developed by volunteers who collaborate through Internet and mostly discussions through online forums and message boards. For example, an OSS environment consists of large number of peripheral users or developers (helping in various tasks except modifying code) which are not present in traditional closed-source software [26].

- There is no central control and planning in OSS [14].

- In general, there is no direct monetary compensation in an OSS environment. In contrast to a commercial setting, developers in OSS environment chose tasks rather than task being assigned to them, can set their own work timings and degree of involvement and can freely join and leave as they are volunteers.

- Lacks many of the traditional mechanisms (plans, system-level design, schedules, and defined processes) used to coordinate software development [18].

The research aim of the work presented in this paper is the following:

- Broad objective: To develop tools and techniques to support a practitioner (offer a real value proposition to a team manager or project leader) in performing risk, threat and vulnerability analysis with respect to a software development team. We refer to risk, threat and vulnerability analysis with respect to (or in context to) people, knowledge and task in an organization (applying ORA tool to an organizational structure).

- Specific objective: In particular, our interest is to investigate social network analysis based techniques for mining software repositories like bug tracking systems to meet the desired objective.

Modern defect tracking systems such as Bugzilla allows a virtual environment in the form of an online threaded discussion for developers to discuss and collaborate with each other for problem solving. We believe that the discussion archives in defect tracking system contain wealth of valuable information that provides new opportunities to study collaboration and interaction between team members. We derive collaboration network from the bug reports in Mozilla Firefox project and investigate the derived social network in the context of risk, threat and vulnerability. The rest of the paper discusses closely related work, lists the claimed novel contributions, describes the experimental dataset, procedure, hypothesis, design decisions and justifications, results and conclusions.

2. RELATED WORK

The work presented in this paper belongs to the subarea of Software Engineering Data Mining or Mining Software Repositories. Mining software repositories is an emerging field that has received significant research interest in recent times. Several tools and techniques based on data mining, text mining and social network analysis based approaches have been proposed in the literature to assist a practitioner in decision making, deriving interesting statistical correlations between different phenomenon and automating software engineering tasks [10] [12].

In general, there is no direct monetary compensation in an OSS environment. In contrast to a commercial setting, developers in OSS environment chose tasks rather than task being assigned to them, can set their own work timings and degree of involvement and can freely join and leave as they are volunteers.

- There is no central control and planning in OSS [14].

- In general, there is no direct monetary compensation in an OSS environment. In contrast to a commercial setting, developers in OSS environment chose tasks rather than task being assigned to them, can set their own work timings and degree of involvement and can freely join and leave as they are volunteers.

- Lacks many of the traditional mechanisms (plans, system-level design, schedules, and defined processes) used to coordinate software development [18].

2.1 SourceForge Projects

Madey et al. collect project and developer data from SourceForge and apply social network analysis to study open source software development phenomenon [14]. Their analysis of the structural properties of the developer collaboration network at SourceForge reveals a power-law model [14].

Ohira et al. apply collaborative filtering and social network analysis techniques on developer and project data collected from SourceForge [19]. They construct three types of networks: developer network, project network and developer-project network with the objective of developing tools to support cross project knowledge collaboration in F/OSS development [19].

Xu et al. apply social network analysis on SourceForge (2003 data dump,) OSS developer and project networks [26]. Their analysis reveals that SourceForge OSS development community is a self-organizing system obeying scale-free property and small world phenomenon [26].

2.2 VA

Luis et al. apply social network analysis to the information in CVS repositories [13]. They derive two types of weighted undirected affiliation networks called as committer network (node is a committer and committers are linked if they have contributed to at least one common module) and module network (node represent a software module and two modules are linked if there is at least one committer who has contributed to both the modules) [13]. Their analysis on the three well-known software projects (Apache, GNOME and KDE) reveals that committers and the modules networks are small-world networks [13].

Huang et al. perform network analysis of version control repository to study grouping structures between developers and modules and compute the relative importance of de-
Table 1: Literature survey of 20 papers (chronological order) on mining software repositories using social network analysis based techniques. SF=SourceForge, VA=Version Archive, SC=Source Code, ML=Mailing List, DTS=Defect Tracking System

<table>
<thead>
<tr>
<th>Study</th>
<th>Repository</th>
<th>Purpose/Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Madey et al., 2002</td>
<td>SF</td>
<td>Testing power-law model in developer collaboration network at SourceForge</td>
</tr>
<tr>
<td>Luis et al., 2004</td>
<td>VA</td>
<td>Testing small-world phenomenon in committers and the modules networks</td>
</tr>
<tr>
<td>Cleidson et al., 2005</td>
<td>SC + VA</td>
<td>Study software artifacts and activities to uncover the structures of software projects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Study collaborative work in large distributed groups such as open source communities</td>
</tr>
<tr>
<td>Ducheneaut et al., 2005</td>
<td>ML + VA</td>
<td>Study relationships OSS newcomers develop over time with social and material aspects of a project</td>
</tr>
<tr>
<td>Huang et al., 2005</td>
<td>VA</td>
<td>Study grouping structures between developers and modules Verify Legitimate Peripheral Participation (LPP) process</td>
</tr>
<tr>
<td>Ohira et al., 2005</td>
<td>SF</td>
<td>Tools to support cross project knowledge collaboration in F/OSS development</td>
</tr>
<tr>
<td>Bird et al., 2006</td>
<td>ML + VA</td>
<td>Examine the relationship between communication and development Correlations between various social network status metrics and source code development</td>
</tr>
<tr>
<td>Crowston et al., 2006</td>
<td>DTS</td>
<td>Empirically distinguishing core group of developers Study size and composition of the core groups</td>
</tr>
<tr>
<td>Howison et al., 2006</td>
<td>DTS</td>
<td>Examine average centralization over time Study stability of participation in project communications</td>
</tr>
<tr>
<td>Sowe et al., 2006</td>
<td>ML</td>
<td>Identification of knowledge experts in open source software projects Study the impact of knowledge brokers and their associated activities in open source projects</td>
</tr>
<tr>
<td>Xu et al., 2006</td>
<td>SF</td>
<td>Testing scale-free property and small world phenomenon in OSS development community Study the effects of co-developers and active users in communication and information flow within the community</td>
</tr>
<tr>
<td>Valetto et al., 2007</td>
<td>ML</td>
<td>Study socio-technical Congruence in development projects</td>
</tr>
<tr>
<td>Bird et al., 2008</td>
<td>ML</td>
<td>Studying sub-communities in Open Source Software (OSS) Projects Study efficiency in the development process, release management and leadership turnover</td>
</tr>
<tr>
<td>Martinez-Romo et al., 2008</td>
<td>VA</td>
<td>Study efficiency in the development process, release management and leadership turnover</td>
</tr>
<tr>
<td>Meneely et al., 2008</td>
<td>VA</td>
<td>Correlation between structure of developer collaboration and product reliability Failure prediction model based on social network analysis of developers</td>
</tr>
<tr>
<td>Pinzger et al., 2008</td>
<td>VA</td>
<td>Correlation between the fragmentation of developer contributions and the number of post-release failures</td>
</tr>
<tr>
<td>Wiggins et al., 2008</td>
<td>DTS + ML</td>
<td>Dynamic analysis of FLOSS team communications across channels</td>
</tr>
<tr>
<td>Sarma et al., 2009</td>
<td>DTS + VA + ML</td>
<td>Study social and technical relationships among different project entities</td>
</tr>
<tr>
<td>Wolf et al., 2009</td>
<td>ML + VA</td>
<td>Examining task-based communication and collaboration in software teams Studying communication pattern to identify causes of build failure</td>
</tr>
<tr>
<td>Meneely et al., 2010</td>
<td>DTS + VA</td>
<td>Improving developer activity metrics Measuring developer collaboration</td>
</tr>
</tbody>
</table>

Pinzer et al. study the correlation between the fragmentation of developer contributions and the number of post-release failures [20]. They derive a developer-module network called as contribution network (modeling associations between developers and software modules or binaries) and use network centrality measures to compute the degree of fragmentation of developer contributions and identify central software modules. Their analysis reveals that central modules are more failure-prone than modules located in surrounding areas of the network [20].

Meneely et al. apply social network analysis to examine human factors in failure prediction [17]. They create a developer network from the update history or change log of files (connect two developers with an undirected edge if the two developers have made a change to at least one file) and demonstrate that developer networks are useful for failure prediction [17].

Martinez-Romo et al. mine data in version control systems to analyze aspects such as efficiency in the development process, release management and leadership turnover [15].

2.3 DTS
Crowston et al. present a sociogram derived from interaction data relating to bug fixing in the SquirrelMail project and demonstrate that active users form a natural buffer between developers and peripheral users. [5]. Crowston et al. present a method to identify the core members of a FLOSS development project. They also analyze several projects and study size and composition of the core groups [5].

Crowston et al. study social structure of FLOSS teams by undertaking social network analysis across time [6]. They demonstrate wide distribution of centralizations across projects and time and present empirical evidences indicating that a change at the center of FLOSS projects is relatively uncommon and participation across the project community is highly skewed [6].

2.4 DTS and VA
Meneely et al. study developer collaboration in issue tracking systems to annotate solution originator and solution approver resulting in an improved developer activity metrics [16]. They apply network analysis techniques and measures to quantify how developers collaborate on projects and demonstrate that many new contributors could be discovered which cannot be revealed by the version control change logs [16].

2.5 ML and VA
Bird et al. create a social network of developers and contributors from the mailing list archives (110,260 messages) of Postgres SQL Server project [2]. They applied standard social network metrics on the derived network and found that developers had higher levels of in-degree, out-degree, and betweenness metrics by at least an order of magnitude over non-developers [2]. They study correlations between various social network status metrics and source code development [2]. Bird et al. perform an empirical analysis on email social network of several projects to study the latent social structure of open-source projects [3].

Wolf et al. apply social-network analysis to gain insight into a software development team’s communication patterns [25]. Their motivation is to investigate social network analysis based techniques that can help to solve software project team collaboration problems. Wolf et al. demonstrate how a project manager can use a social network to identify a communication broker between two project members and investigate correlation between properties of an integration team’s social network and code integration outcome [25].

Sowe et al. analyze Debian lists to investigate the impact of knowledge brokers (people who bridge the gap between expert software developers and user communities) and their associated activities in open source projects [23]. They apply social network analysis to visualize affiliation between various participants in the lists and demonstrate that the proposed methodology can be used to help identify active and valuable expert human resources [23].

Ducheneaut et al. study relationships OSS newcomers develop over time with social and material aspects of a project [9].

2.6 ML
Valetto et al. study socio-technical software networks to analyze communication interactions between stakeholders and inter-relationships between artifacts, to compute congruence [24]. They mine mailing lists and bulletin boards and describe a method to measure the degree of alignment between social relationships and software relationships [24].

2.7 ML and DTS
Wiggins et al. study communication patterns in two projects (instant messaging clients Fire and Gaim) [1]. They mine data present in email lists, forums, and trackers were study various aspects like communication dynamics and communication centralization trends across the venues within each project. They introduce a method for intensity-based smoothing in dynamic social network analysis and demonstrate that venues in both projects trended toward decentralization over time [1].

2.8 DTS, VA and ML
Sarma et al. describe a socio-technical dependency browser that to enable exploration of various relationships between software artifacts, developers, bugs, and communications [21]. They develop a tool called as Tesseract and demonstrate its value for new developers or managers in creating a mental map of the project [21].

2.9 SC and VA
Cleidson de Souza et al. study relationship between software artifacts and software development processes and present a visualization-based approach for analyzing software projects [8]. They apply network analysis to examine various aspects: relationship between members of the development team and code contributions, study core and periphery divisions and distinguish between various forms of peripheral participation, core-periphery shifts and authorship changes [8].

3. RESEARCH GAP AND NOVEL CONTRIBUTIONS
Table 1 presents a summary of our literature review on the application of social networking analysis for mining software
We synthesize the existing work and identify a research gap in the area of mining software repositories using social network analysis based techniques. Based on our close and careful study of existing literature, we conclude that mining software repositories in the context of risk, threat and vulnerability analysis is a relatively unexplored area. The research objective of this paper is to throw light on this relatively unexplored domain. To the best of our knowledge, Table 1 presents the first systematic survey and classification of papers (20 research papers categorized and listed in a chronological order from 2002 to 2010) focused on the topic of mining software archives using social network analysis. We make the following unique contributions in context to related work:

1. While there has been work on applying Social Network Analysis (SNA) techniques on software repositories for various research purposes (refer to Table 1), the application of SNA techniques to specifically study project risk, threat and vulnerability (for an open source software environment) in a focused and in-depth manner is a novel contribution of this work. To the best of our knowledge, this is the first study in the literature in the sub-field of mining software repositories that addresses an important issue of mining defect tracking systems (software archive containing historical project information) for risk and threat analysis.

2. While there has been work on mining structured and unstructured data in defect tracking system for solving various software engineering tasks, the analysis of defect tracking system to study developer collaboration and interaction data is relatively unexplored. Based on our literature survey, the only work that we have come across on the topic of mining collaboration data in defect tracking systems are the papers by Crowston et al. [6] [5]. While this paper and the work by Crowston et al. has similarity in terms of the application of social network analysis to study collaboration and interaction data in bug tracking systems, there are several noticeable differences. The method to derive the collaboration network by Crowston et al. and this paper are different. Crowston et al. connect two developers with an edge if one of the developers is a sender of the message and the other is the preceding sender whereas we extract all the unique developers (irrespective of their order of message sending) for a bug report and connect all of them with each other with an edge. Our rationale and belief is that since all the developers collaborated towards a common goal of bug fixing, they share a tie or relationship. Also, the focus of the work by Crowston et al. is to study methods to identify the core members of a FLOSS development project (and study size and composition of the core groups), the perspective of this paper is more towards risk, threat and vulnerability analysis. While this paper is not the first study on deriving a network from collaboration data in bug tracking systems (Bugzilla bug databases), the number of edge connecting two nodes if the developers representing the nodes collaborated towards a common goal of bug fixing, they share a tie or relationship. The rationale is that a link represents collaboration (developers working together or interacting with each other to achieve a common goal or agenda and have shared interest or expertise). The same process is applied for other bug reports. Figure 1 shows that there are four developers (Dan, David, Hans and Stephen) who collaborated and discussed with each other towards resolving a given issue or bug. We create four nodes labeled as Dan, David, Hans and Stephen and connect all of them with each other through undirected (symmetrical relationship) edges. The rationale is that a link represents collaboration (developers working together or interacting with each other to achieve a common goal or agenda and have shared interest or expertise). The same process is applied for other bug reports in the experimental dataset (Bugzilla bug databases for Mozilla Firefox project). The degree of a node is equal to the number of edges connected to the node and the weight of an edge connecting two nodes represents the number of times the developers representing the nodes have collaborated with each other. The weight denotes the strength of a connection or tie. All the network visualization and graphs in the fol-

3. We present several perspectives in context to risk, threat and vulnerability analysis. We use standard and widely used Social Network Analysis [7][4][22] based tools and techniques to mine collaboration data implied in the defect tracking system (Bugzilla) of Mozilla Firefox project. We present empirical results and insights gained from mining real-world data available in public domain.

4. RESEARCH METHOD, EXPERIMENTAL DATA, ANALYSIS AND RESULTS

4.1 Developer Collaboration Network

A collaboration network is modeled as an undirected graph in which the nodes represent developers and edges represent ties between developers. We create a link between two nodes if the developers representing the nodes collaborated with each other in resolving a bug. Consider Figure 1 which presents a snapshot of a Mozilla Firefox (popular and widely used web-browser) bug report. The bug report in Figure 1 consists of structured data fields such as the bug id, title, product, component, severity, priority, reporting time and assigned-to. The bug report also consists of a threaded discussion consisting of developer comments (along with the developer id and time-stamp). Figure 1 shows that there are four developers (Dan, David, Hans and Stephen) who collaborated and discussed with each other towards resolving a given issue or bug. We create four nodes labeled as Dan, David, Hans and Stephen and connect all of them with each other through undirected (symmetrical relationship) edges. The rationale is that a link represents collaboration (developers working together or interacting with each other to achieve a common goal or agenda and have shared interest or expertise). The same process is applied for other bug reports in the experimental dataset (Bugzilla bug databases for Mozilla Firefox project). The degree of a node is equal to the number of edges connected to the node and the weight of an edge connecting two nodes represents the number of times the developers representing the nodes have collaborated with each other. The weight denotes the strength of a connection or tie. All the network visualization and graphs in the fol-

Figure 1: Illustrative example to demonstrate construction of Collaboration Network from Threaded Discussions in Bug Reports
lowing sections are generated using two tools: ORA³ and Pajek⁴. Figures 2 to 12 are generated using ORA and Figures 13 to 16 are generated using Pajek.

4.2 Core, Semi-Periphery and Periphery Stratification/Characteristics

We hypothesize that the collaboration network follows a core-periphery pattern. A core-periphery pattern consists of two classes of nodes: core and periphery. The nodes belonging to the class of core are the dominant and well connected nodes whereas the nodes belonging to the class of peripheral nodes have fewer connections and importance. Borgatti et al. formalize the concept of a core-periphery structure in a network and propose algorithms and statistical tests for identifying this structure [Borgatti2000]. Borgatti et al. mention that the core-periphery pattern has been prevalent in diverse fields such as world system, organizational system and scientific citation networks [Borgatti2000]. The more closely related work to the stated research question is the work done by Crowston et al. who study core and periphery in Free/Libre and Open Source software team communications [6][5]. Crowston et al. observe that normally the core group of developers is small (3-10 is adequate for most of the projects) [6][5]. We perform a visual and numerical analysis on the dataset to test our a-priori hypothesis of the presence of core-periphery phenomenon. Figures 2 and 3 presents a circular layout in the form of rings. A node representing a developer is placed in a ring which is proportional to the developer’s degree or number of connections with other developers. The core developers are placed in the center or the innermost ring and the peripheral developers are shown in the outermost ring or in the margin. Figures 2 and 3 shows a clustering or partition of vertices into three broad types of classes: core, semi-periphery, and periphery. The three subsets are mutually exclusive. The developers who have worked with many other developers and are well connected form the core. Identification of core developers and peripheral developers is an important problem in software projects [6][5].

A visual inspection of Figure 2 and 3 indicates presence of a core-periphery pattern. We notice that there are a small number of developers in the core having high degree centrality. There are several developers in semi-periphery and periphery. It can be useful for a project manager to understand the percentage of developers in the core, semi-periphery, and periphery and identify core developers from risk analysis perspective. We identify the most central (prominent or important as their absence can have negative consequences or weaknesses or vulnerabilities. For example, knowledge of developers acting as bridges or gatekeepers is important as their absence can have negative consequences or

4.3 Developer to Bug Severity Relationship

Figure 4 presents a two mode network depicting relationship between bug severity and developers. Our objective is to answer questions like "who are the developers working on critical bugs", "are there developers working on a variety of bug severity and are there developers working only on bug reports belonging to a certain bug severity". Revealing hidden patterns present in the bug severity and developer network can be useful to a project manager as identifying developers working intensively on major or critical bugs (which are blockers) is important from Risk and Vulnerability perspective. Figure 4 highlights (small spots in dark color in the center) central developers. Figure 4 reveals non-trivial insights and answers the stated questions. The developers working only on certain type (severity level) of bugs and developers working on multiple types of bugs can be detected.

4.4 Authority Centrality, Betweenness Centrality and Knowledge Exclusivity

Figures 5,6,7,8,9 and 10 throw light on the aspects of authority centrality, betweenness centrality and knowledge exclusivity. The bar chart for authority centrality and betweenness centrality is derived from the developer to developer network whereas the knowledge exclusivity bar chart is derived from developer to component network (analyzing 500 bug reports for illustrative purposes). The bar charts reveals ranking of top 20 developers based on their centrality measures and the difference between their centrality scores. The scatter plot reveals that the developers who have high authority centrality does not mean that they have high betweenness centrality or knowledge exclusivity also. Answers to questions such as "who is the most connected developer", "which developers are communication bridges or brokers", "which developers are in the 2nd or 3rd tier of leadership", "what is the extent of difference between various centrality scores of developers" can be useful to a project manager to understand the organizational structure and its strengths as well as weaknesses or vulnerabilities. For example, knowledge of developers acting as bridges or gatekeepers is important as their absence can have negative consequences or

³http://www.casos.cs.cmu.edu/projects/ora/
⁴http://pajek.imfm.si/doku.php
⁵http://www-archive.mozilla.org/about/staff
⁶http://www.mozilla.org/hacking/reviewers.html

Figure 4: A two mode network depicting relationship between bug severity and developers

Figure 4 presents a two mode network depicting relationship between bug severity and developers. Our objective is to answer questions like "who are the developers working on critical bugs", "are there developers working on a variety of bug severity and are there developers working only on bug reports belonging to a certain bug severity". Revealing hidden patterns present in the bug severity and developer network can be useful to a project manager as identifying developers working intensively on major or critical bugs (which are blockers) is important from Risk and Vulnerability perspective. Figure 4 highlights (small spots in dark color in the center) central developers. Figure 4 reveals non-trivial insights and answers the stated questions. The developers working only on certain type (severity level) of bugs and developers working on multiple types of bugs can be detected.

4.4 Authority Centrality, Betweenness Centrality and Knowledge Exclusivity

Figures 5,6,7,8,9 and 10 throw light on the aspects of authority centrality, betweenness centrality and knowledge exclusivity. The bar chart for authority centrality and betweenness centrality is derived from the developer to developer network whereas the knowledge exclusivity bar chart is derived from developer to component network (analyzing 500 bug reports for illustrative purposes). The bar charts reveals ranking of top 20 developers based on their centrality measures and the difference between their centrality scores. The scatter plot reveals that the developers who have high authority centrality does not mean that they have high betweenness centrality or knowledge exclusivity also. Answers to questions such as "who is the most connected developer", "which developers are communication bridges or brokers", "which developers are in the 2nd or 3rd tier of leadership", "what is the extent of difference between various centrality scores of developers" can be useful to a project manager to understand the organizational structure and its strengths as well as weaknesses or vulnerabilities. For example, knowledge of developers acting as bridges or gatekeepers is important as their absence can have negative consequences or
4.5 Component Severity and Component Developer Relationship

Figure 11 presents a 2-mode graph consisting of two types of nodes. The node with the triangle shape represents severity and the node with a round spot represents components. The objective is to understand the relationship between components and severity and extract patterns useful to a project manager or the team. Identification of components which have fault-prone or components which have high density of critical bugs can be useful from the perspective of risk and threat analysis. A project manager can direct manpower resources and expertise according to the needs based on error-proneness of a module or component. This information is non-trivial to acquire explicitly and in this paper we demonstrate how such implied patterns can be uncovered from static record keeping software archives such as the bug database. For example, in Figure 11, we observe that there are certain components which have received only critical bugs (for a pre-defined time window). The components having only critical bugs are labelled in the figure. Also, notice that the font size of the component labels are in the order of their frequency of bugs, i.e., the component Networking Cookies is having maximum number of critical bugs.

Figure 12 presents a 2-mode graph in which one of the node type is developer and the other node type is component. Our objective is to extract interesting patterns and actionable information that can support a practitioner in decision making with respect to developer (people) and component (product) interaction. For example, identifying developers that have knowledge of varied components in contrast to developers who are expert in one or two specific components can be useful to a project manager. Knowledge of the number of developers working on each component can also be useful for planning activities. In Figure 12, the square nodes (representing components) are sized by degree. This graph in Figure shows that there are some developers (in the centre of the graph) who are working on multiple components. We notice that the developer
Figure 7: Scatter plot between authority centrality (x-axis) and betweenness centrality (y-axis) score of developers. The points in the graph represents developers in the sample dataset. The graph reveals that there are certain developers who have high betweenness centrality but not high authority centrality and vice-versa. This is an important and non-trivial insights for a project manager as understanding the position of a developer in the network from multiple perspective can be useful for decision making (such as identification of gate-keepers, knowledge brokers, important and critical developers).

Figure 8: Scatter plot between authority centrality (x-axis) and knowledge exclusivity (y-axis) score of developers. The point in the graph represents developers in the sample dataset. The graph reveals that there are certain developers who have high authority centrality but not high knowledge exclusivity score and vice-versa. In contrast to Figure 7, the correlation between the two variables under study in Figure 8 is less. We believe that it is non-trivial to derive such information explicitly (in a constantly changing and dynamic environment) and such information is an important input to a managers de-risking strategy formulation.

Figure 9: Top 20 developers in terms of exclusivity knowledge score. The y-axis represents the knowledge exclusivity scores. The graph reveals that the order or ranking of the Top 20 developers in terms of various dimensions such as betweenness centrality, authority centrality and knowledge exclusivity are not exactly the same. It is not necessary that a developer who is a gate keeper or a bridge is the developer with maximum connection or is the developer who has exclusive knowledge.

Figure 10: Scatter plot between betweenness centrality and knowledge exclusivity score of developers
with the id bzbarsky@mit.edu.in has worked on maximum number of components (= 54 in this case). This signifies the importance and expertise of that particular developer. The graph reveals components on which a significant number of developers are working. We notice that the component Layout has 84 developers worked on it. The developers on the periphery are the ones who have worked on only one component whereas the developers in the centre are the ones who have participated in bug-fixing activity of multiple components.

4.6 Clusters and Cohesive Subgroups

We study the developer to developer network of Figure 13 and component to component network (the component-component network created from an adjacency matrix that is derived from the incidence matrix between developers and components) shown in Figure 16 to identify cohesive subgroups and teams or community. Figures 13 and 16 shows the position of the developer and component within their respective networks. A sub-group is a set of nodes, developers or entity (as a node can represent a component also) who interact or collaborate with each other more frequently or intensely (such that the weights of the edges between the nodes represents the intensity of collaboration) than nodes outside the group. Figure 14 displays a degree partition of developers. Figure 14 has 556 nodes and 25 partitions based on the degree. For example, we observe that there are 9 developers of degree 10 which constitutes 1.61% of all the developers. Similarly, we notice that there are 15, 14 and 18 developers with degrees 5, 6 and 7 respectively whereas there are 2, 1 and 1 developer with high degrees of 27, 29 and 30 respectively. Figure 14 helps a project manager in understanding the position of each developer in the community and the degree of centralization or distribution for the project. Figure 14 is generated using Pajek’s degree partition feature whereas Figure 15 is generated using Pajek’s core partition feature. There are 5 clusters in Figure 15 of frequency 43.88%, 17.98%, 12.94%, 8.45% and 16.72% respectively.

5. DISCUSSION

We believe that understanding the interaction between developers and between developers and software product can reveal useful insights pertaining to risk, threat and vulnerability analysis in an organization. Static record keeping databases such as a defect tracking system provides opportunity to derive hidden and interesting patterns useful to a project manager and developers. We apply standard social network analysis approaches to investigate collaboration network derived from an issue tracker and argue in support of our hypothesis using illustrative examples and empirical results that social network analysis can play a crucial role in risk, threat and vulnerability analysis of an organizational structure. In this paper we throw light on aspects like core-periphery patterns, cohesive subgroups and clusters, centrality (authority and bridges), patterns present in various two mode networks such as developer to component, developer to severity and component to severity. We also present a systematic survey of the previous work in the area of social network analysis for mining software repositories.

Figure 11: Component-Severity relationship

Figure 12: Component-Developer relationship

Figure 13: A snapshot of developer to developer network

Figure 14: A degree partition of developers

Figure 15: A snapshot of component to component network

Figure 16: A snapshot of component to component network
6. REFERENCES


