Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2010, Article ID 273080, 9 pages
doi:10.1155/2010/273080

Research Article

On the Use of Issue Tracking Annotations for Improving Developer Activity Metrics

Andrew Meneely and Laurie Williams

North Carolina State University, 890 Oval Drive, Engineering Building 2, Room 3272, Campus Box 8206, Raleigh, NC 27695-8206, USA

Correspondence should be addressed to Andrew Meneely, apmeneel@ncsu.edu

Received 5 September 2010; Revised 2 December 2010; Accepted 31 December 2010

Academic Editor: Giulio Concas

Understanding and measuring how teams of developers collaborate on software projects can provide valuable insight into the software development process. Currently, researchers and practitioners measure developer collaboration with social networks constructed from version control logs. Version control change logs, however, do not tell the whole story. The collaborative problem-solving process is also documented in the issue tracking systems that record solutions to failures, feature requests, or other development tasks. We propose two annotations to be used in issue tracking systems: solution originator and solution approver. We annotated which developers were originators or approvers of the solution to 602 issues from the OpenMRS healthcare system. We used these annotations to augment the version control logs and found 47 more contributors to the OpenMRS project than the original 40 found in the version control logs. Using social network analysis, we found that approvers are likely to score high in centrality and hierarchical clustering. Our results indicate that our two issue tracking annotations identify project collaborators that version control logs miss. Thus, issue tracking annotations are an improvement in developer activity metrics that strengthen the connection between what we can measure in the project development artifacts and the team's collaborative problem-solving process.

1. Introduction

The quality of many software products rests on teams of people who are collaborating with each other. Measuring how teams of developers collaborate on software projects can provide valuable insight into the software development process. One class of metrics, called *developer activity metrics*, analyzes the structure of a development team by quantifying how developers collaborate with each other [1]. Already, developer activity metrics have been shown to predict failures [2, 3], predict vulnerabilities [1], and provide insight on individual projects [4–7].

Many studies using developer activity metrics use version control change logs to determine who is working on which part of the system. Version control change logs, however, do not tell the whole story. While a version control system records the final solution to an issue, the elements of how that solution came to be are captured in other artifacts, such as the issue tracking system. In issue tracking systems, developers can take a known issue (e.g., a failure or a feature request) and assign it a "ticket". Developers can then link pertinent artifacts to that ticket, such as patches, change sets in the version control system, error logs, or screenshots. Also on the ticket is an online discussion of how to resolve the issue.

Consider the following scenario (taken from http://dev.openmrs.org/ticket/1171). A large open source project has an open ticket that needs resolving. A user named Frank then submits a patch to fix the problem by linking his patch to the ticket. An online discussion amongst the system's developers and users then takes place, which ends in developer Ben deciding that Frank's solution is correct. Ben then applies the changes from Frank's patch, linking the version control change set to the ticket. The version control change logs would show that only Ben has fixed the problem when, in fact, Frank *originated* the solution, while Ben *approved* the solution. That the two developers collaborated on a solution ought to be captured in developer activity metrics.

We propose two issue tracking ticket annotations: solution originators and solution approvers. Intended to "give credit where credit is due", these annotations can be used to augment the logs from version control systems to provide more accurate information about who contributed to which parts of the system.

The objective of this research is *to improve the information gained by measurements of developer collaboration by introducing and analyzing two issue tracking annotations: solution originator and solution approver*. Our aim is to evaluate the usefulness of the annotation in terms of discovering collaborators in a software development project. We examined the online discussions of 602 tickets from the OpenMRS (http://openmrs.org/) healthcare web application issue tracking system, annotating which developers were originators and/or approvers of the solution to the ticket.

We performed an empirical analysis of how much information was gained by using issue tracking annotations in combination with version control change logs. Forming social networks of developers based on our data, we examined correlations between the annotations and several social network analysis metrics. Our social network analysis metrics come from centrality and hierarchical clustering techniques.

The rest of the paper is organized as follows. Section 2 describes background and related work with respect to developer activity metrics, centrality, developer networks, and issue tracking systems. Section 3 describes the developer network. Section 4 describes our data collection process. Section 5 describes our analysis and results. Sections 6 and 7 summarize our limitations and conclusions.

2. Background and Related Work

As a project progresses, developers make changes to various parts of the system. With many changes and many developers, changes to files tend to overlap: multiple developers may end up working on the same files around the same time, indicating that they share a common contribution, or a *connection*, with another developer. Some developers end up connected to many other highly connected developers, some end up in groups (clusters) of developers, and some tend to stay peripheral to the entire network.

In this paper, we use network analysis to quantify how developers collaborate on projects. Network analysis is the study of characterizing and quantifying network structures, represented by graphs [8]. In network analysis, vertices of a graph are called nodes, and edges are called connections. A sequence of nonrepeating, adjacent nodes is a path, and a shortest path between two nodes is called a geodesic path (note that geodesic paths are not necessarily unique). Informally, a geodesic path is the "social distance" from one node to another.

"*Centrality*" metrics are used to quantify the location of a node relative to the rest of the network. In this study, we use two measures of node centrality: degree and betweenness. The "*degree*" of a node is equal to the number of neighbors a developer has in the network. While the "*degree metric*" is based on direct connections to other developers, the "*betweenness*" metric is based on a developer's indirect connections to the rest of the network. If a developer has many connections, the betweenness [8] of node *n* is defined as the number of geodesic paths that include *n*. A high betweenness means a high centrality.

"*Clustering*" algorithms are techniques for detecting community structures in social networks. A cluster of nodes is a *set* of nodes such that the number of intraset connections greatly outnumbers the number of interset connections [8]. A cluster of developers, then, has more connections within the cluster than to other developers. Furthermore, since communities can have many layers of subcommunities, "*hierarchical clustering*" techniques detect the clusters within clusters. In this study we use the Girvan-Newman [8] hierarchical clustering algorithm, which applies centrality concepts to edges in the network and iteratively generates a hierarchy of developers according to the number and size of the clusters they belong to.

Also, we use the term "*issue*" to include all potential types of change requests in a system, including failures, vulnerabilities, feature requests, and tasks. When we refer to a "*ticket*," we are referring to the record of a specific issue as kept by the issue tracking system. Tickets can have discussions on them, in a message board fashion. The solution to a ticket is embodied in a "*change set*," which is a set of changes made to a code base kept track of by the version control system.

Much work has been done in the area of measuring developer collaboration on software projects. The applications of these measurements are particularly diverse, ranging from failure and vulnerability prediction to studying open source software projects.

Gonzales-Barahona et al. [6] were the first to propose the idea of creating developer networks as models of collaboration from version control systems. The authors' objective was to present the developer network and to differentiate and characterize projects. Their work did not include any integration with issue tracking systems.

Bird et al. [5] used developer networks to examine social structures in open source projects. Discussing the bazaar-like development of open source projects, the authors empirically examine how open source developers self-organize. The authors use similar network structures as our developer network to find the presence of subcommunities within open source projects. In addition to examining version control change logs, the authors mined email logs to find a community structure. The authors conclude that subcommunities do exist in open source projects, as evidenced by the project artifacts exhibiting a social network structure that resembles collaboration networks in other disciplines.

Pinzger et al. [3] proposed a similar structure to the developer network, called the contribution network. The contribution network is designed to use version control data to quantify the direct and indirect contribution of developers on specific resources of the project. The researchers used metrics of centrality in their study of Microsoft Windows

Vista and found that closeness was the most significant metric for predicting reliability failures. Files that were contributed to by many developers, especially by developers who were making many different contributions themselves, were found to be more failure-prone than files developed in relative isolation. The finding is that files which are being focused on by many developers are more likely to have a failure than files developed by few developers.

Meneely et al. [2] examined the relationship between developer activity metrics and reliability. The empirical case study examined three releases of a large, proprietary networking product. The authors used developer centrality metrics from the developer network to examine whether files are more likely to have failures if they were changed by developers who are peripheral to the network. The authors formed a model that included metrics of developer centrality, code churn (the degree to which a file was changed recently), and lines of code to predict failures from one release to the next. Their model's prioritization found 58% of the system's failures in 20% of the files, where a perfect prioritization would have found 61%. The study did not include integration with issue tracking systems.

Sarma et al. [9] built a tool, called Tesseract, to visualize developer connections within a software project. Designed to build a useful summary of what they call the "network of artifacts" in each software project, Tesseract gathers its information from version control change logs, source code dependency graphs, issue tracking tickets, and developer communication logs. In this particular study, they established that Tesseract was both a usable and useful tool according to the user studies and interviews they conducted.

Nagappan et al. [10] created a logistic regression model for failures in the Windows Vista operating system. The model was based on what they called "Overall Organizational Ownership" (OOW). The metrics for OOW included concepts like organizational cohesiveness and diverse contributions. The authors found that more edits made by many noncohesive developers lead to more problems post release. The OOW model was able to predict with 87% average precision and 84% average recall. The OOW model bears a resemblance to the contribution network by Pinzger et al. [3] in that both models attempt to differentiate healthy changes in software from the problematic changes.

Meneely and Williams [1] applied developer activity metrics to security data in examining aspects of the saying "Many eyes make all bugs shallow" (known as Linus' Law [11]). Using both developer networks and the contribution networks proposed by Pinzger et al. [3], the authors examined several metrics related to developer collaboration, including a clustering metric applied to developer networks. The authors found that files changed by nine or more developers were 16 times more likely to have at least one vulnerability than files changed by fewer than nine developers. Their analysis did not include data from issue tracking systems, only from version control change logs.

Lastly, our original proposal for extending developer networks using issue tracking annotations included an analysis of the OpenMRS system [7].

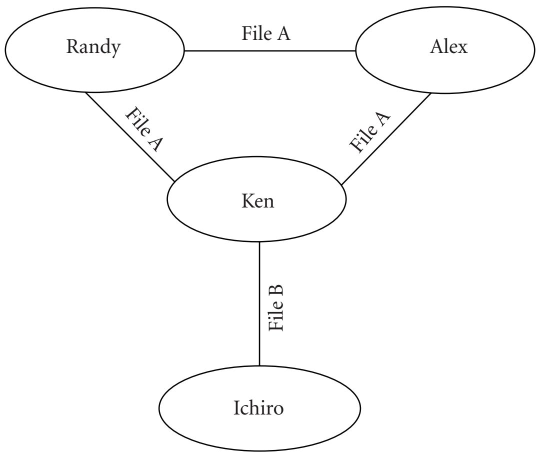


Figure 1: Resulting developer network from Table 1.

Table 1: Example developer contributions.

Developer	Contributions
Alex	File A
Ken	File A, File B
Randy	File A
Ichiro	File B

The developer network from Table 1 is shown in Figure 1.

3. Extending Developer Networks

The developer network is designed to represent the complex structure of development in terms of people. The idea is to infer "who is working with whom" by examining "who is working on the same code". In our developer network, developers are represented as nodes, and edges exist between two nodes where two developers made contributions to the same source code file within one month. As a result, the developer network is an undirected, unweighted, and simple graph.

In prior research [1–3, 5, 6, 10], the existence of a "contribution" to a given source code file was gathered from only version control change logs. We will refer to such a structure as a "*developer network*" (or "regular" developer network).

For example, suppose that we have the contribution table found in Table 1.

In Figure 1, developer Ken has a degree of three. Ken also has a betweenness of five since he is on five of the geodesic paths (three originating from himself). Ichiro has a betweenness of three, while Randy and Alex have a betweenness of four. More examples of developer networks and their usage can be found in other works [1, 2, 5, 9].

In this study, we extend the notion of contributions beyond version control change logs to originators and approvers of solutions in issue tracking systems. Thus, if a person was found to be the originator or approver of the solution to a ticket, that person would be marked as making a contribution to the source code files in the final commit (more information can be found in Section 4).

Therefore, we will refer to the developer network gathered from version control logs *and* our issue tracking annotations as the *extended developer network*.

For this study, we annotated tickets manually by examining issue tickets post hoc, as described in Section 4. However, this data could be gathered earlier and with less labor if the issue tracking system supports a "solution approver" and a "solution originator" field on the ticket. If issue tracking systems allowed for our two annotations, the developers on each ticket could decide who are the solution originators and solution approvers and could form an extended developer network automatically. To our knowledge, no such field exists in issue tracking systems such as Bugzilla (http://www.bugzilla.org/) or Trac (http://trac.edgewall.org/).

4. Collecting Annotations

The developers of the OpenMRS project use Trac for their issue tracking system and Subversion (http://subversion.tigris.org/) (SVN) for their version control system. By default, Trac will link a coded change set in SVN to a ticket if the developer uses the hash mark (#) and the ticket number in the Subversion commit message. While this feature is optional, we found that the OpenMRS developers were meticulous about linking change sets to tickets when the issue was resolved. OpenMRS had over 1900 tickets logged during the time that we studied, of which 602 resulted in a change set to resolve the issue.

We manually examined the discussions of those 602 tickets to annotate who on the ticket were the approver and originator of the solution to the issue. We used the following steps for each ticket.

(1) Read the ticket's description to understand the problem.

(2) Read the online discussion directly on the ticket.

(3) Read any discussions in separate forums (e.g., mailing list) linked from the ticket.

(4) Read the SVN comments left by the person who committed the solution to version control.

(5) Compare the patches attached to the ticket to the solution committed to SVN and determine which patches were used in the issue's solution.

(6) Based on the information gained in steps (1)–(4), decide who the originators are and who the approvers are.

The first author and an additional researcher executed the latter six steps independently. Both researchers then compared annotations and resolved each disagreement in annotation.

We considered a person to be an originator if

(i) a person submitted a patch that was a major part of the solution to the issue, or

(ii) a person introduced code (e.g., wrote code directly into the ticket's message board), and that code was used in the solution, or

(iii) the committer of the solution attributed someone on the ticket in their Subversion commit message.

We considered a person to be an approver if

(i) the person changed the ticket status to "approved", or

(ii) the person, in the course of the discussion, made the final decision as to what ought to be done in the code (but did not necessarily enact the change in the code), or

(iii) the person applied the patch and closed the ticket.

One ticket could have multiple originators and multiple approvers. Not included in our annotations are people who made contributions to the ticket discussion but did not participate in the final solution. Furthermore, a person could be both an originator and an approver to a solution if they created a public ticket and then resolved it themselves. Lastly, a person could be neither an originator nor an approver but still make contributions to the code if they only committed code directly to version control system without participating in any solutions to public tickets. We call these people "contributors".

If a person was an originator or approver, they were recorded as making a contribution to the code affected by the solution at the time the solution was applied in addition to the original committer to the version control system. With the records of contributions from contributors, approvers, and originators, the developer network was calculated as described in Section 2. As a result of this construction, originators and approvers of the same ticket are always connected to each other. This automatic connection matches the notion of a collaboration connection since originators and approvers are collaborating on a solution.

5. Evaluation

In this section, we examine whether information about developer collaboration can be gained by analyzing two issue tracking annotations: solution originator and solution approver. First, in Section 5.1, we examine the developers found by applying issue tracking annotations. Next, in Sections 5.2 through 5.5, we evaluate the following scientific hypotheses.

H_1: *Approvers have a higher developer network centrality than nonapprovers.*

H_2: *Originators have a higher developer network centrality than nonoriginators.*

H_3: *Central developers in regular developer networks are also central in extended developer networks.*

H_4: *Approvers belong to more clusters than nonapprovers.*

5.1. Analyzing Information Gained from Annotations. First, we must ask if our anecdotal observations of SVN logs are true: do the originator and approver annotations provide more information that was not gained from analyzing the version control change logs? Perhaps the commits from the

Table 2: Approvers in regular and extended developer networks.

	Only in regular DN	Only in extended DN	Total
Nonapprover	24 (60%)	**39 (82%)**	63 (72%)
Approver	16 (40%)	8 (18%)	24 (27%)
Total	40 (100%)	47 (100%)	87 (100%)

Table 3: Originators in regular and extended developer networks.

	Only regular DN	Only in extended DN	Total
Nonoriginator	13 (32%)	7 (15%)	20 (23%)
Originator	27 (68%)	**40 (85%)**	67 (77%)
Total	40 (100%)	47 (100%)	87 (100%)

version control system are enough to identify the entire structure of the team.

A visual comparison of the developer network and extended developer network can be found in Figure 2 (shown in the appendix). The developer network more than doubled in size when including annotations. The developer network turned out to have 40 developers, while the extended developer network had 87 developers. As a result, our annotations identified 47 developers who did not make SVN commits but contributed to a solution adopted into the project.

Furthermore, the majority of the 47 developers found only in the extended network were originators and not approvers. Tables 2 and 3 show how the developer counts break down in terms of annotations in each network. Had one used a developer network from only version control logs, one might conclude that the development team is 40% approvers, when in fact there are fewer approvers (27%).

The high rate of nonapprovers is found only in the extended network (cell highlighted in gray). Also, Table 3 shows a high rate of originators found only in the extended network (cell highlighted in gray).

Additionally, the complexity of the network structure increased dramatically. Figure 2 (shown in the appendix) visually demonstrates the difference between the two networks rather starkly. Both networks have the same layout; so developers in both networks are in the same location in each diagram.

Therefore, when comparing the regular developer network to the extended developer network, the number of contributors to solutions adopted by the OpenMRS project more than doubles in size because many originators and nonapprovers are being accounted for.

5.2. Approver Centrality. Developer network centrality is a measure of how directly and indirectly a developer is to the rest of the network. When a developer has a high centrality, then he or she has worked on files with many other people. In previous work [2], we found that developer centrality can be used for predicting failures in files.

Table 4: Approver centralities.

Metric	Nonapprover mean	Approver mean	MWW P-value < .01?	Power
Degree	6.0	16.9	Yes	0.81
Betweenness	2.7	132.6	Yes	0.99

Furthermore, we observed that developers we annotated as solution approvers tended to be people who were well-known and knowledgeable enough to be trusted with approving solutions to problems. Therefore, we hypothesize that developer centrality is correlated with being an approver.

H_1: *Approvers have a higher developer network centrality than nonapprovers.*

We evaluate H_1 by evaluating the following null and alternative hypotheses.

$H_{1(0)}$: *Approvers have the same developer network centrality as nonapprovers.*

In this paper, we use two measures of centrality: degree and betweenness (defined in Section 2). We used the Mann-Whitney-Wilcoxon (MWW) test (at $P < .05$) to examine the differences in centrality between approvers and nonapprovers. We chose the nonparametric MWW because it does not assume that developer centralities are normally distributed. To evaluate which group has the higher centrality, we compared the means. Since we are evaluating multiple hypotheses, we used a Bonferroni correction; so we check our P-values against .01 instead of the traditional .05. Lastly, we perform a power analysis to ensure that our sample groups are large enough and compare the results against the traditional 8. Table 4 shows our results.

For both metrics, the approvers had a higher developer centrality than nonapprovers; thus we reject the null hypothesis $H_{1(0)}$. Therefore, central developers of the extended developer network are also approving solutions to the issue tickets. Figure 3 (shown in the appendix) visually illustrates that the approvers are generally well-connected developers in the network.

5.3. Originator Centrality. Centrality is related not only to *how many* connections a developer has, but also *to whom* a developer is connected (e.g., if a developer is connected to a very central developer, she becomes more central herself). Since originators and approvers on the same ticket are automatically connected to each other, then if approvers are central, then originators are also central. Thus, we examine the following hypothesis.

H_2: *Originators have a higher developer network centrality than nonoriginators.*

The null version of this hypothesis is as follows.

$H_{2(0)}$: *Originators have the same developer network centrality as nonoriginators.*

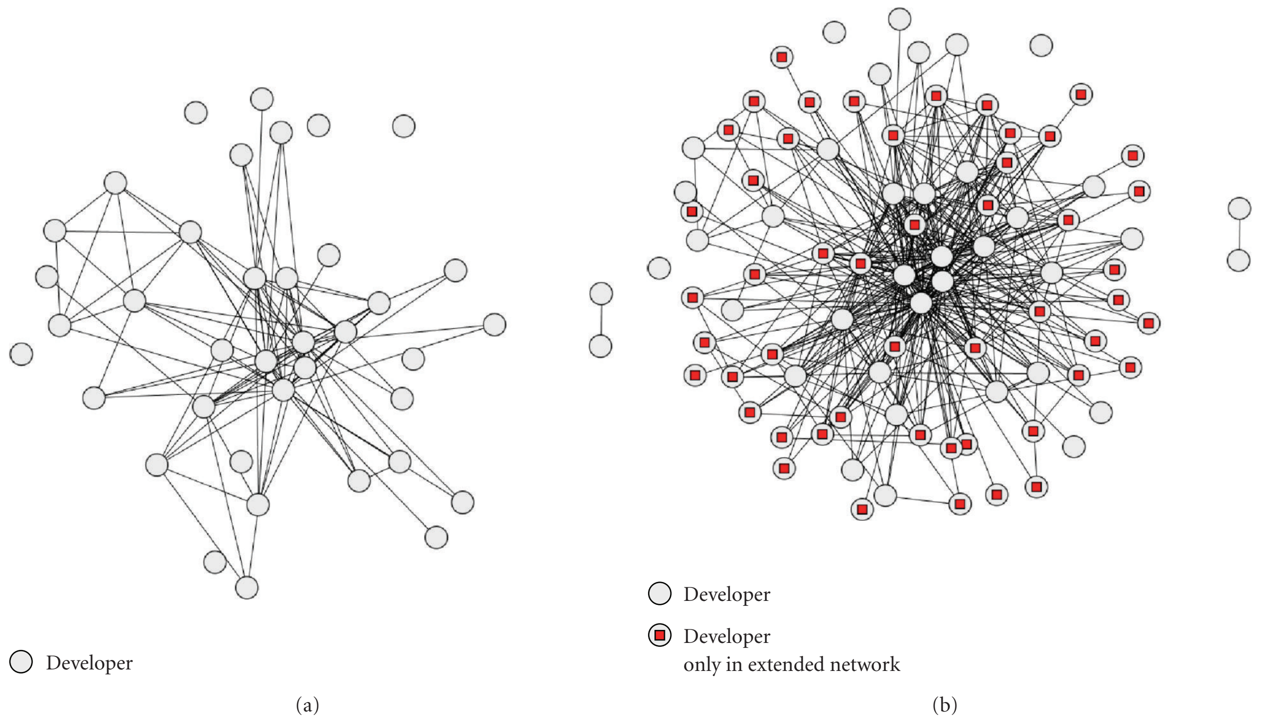


Figure 2: (a) Regular developer network and (b) extended developer network of the same layout.

We applied the same analysis as in the previous section. Our results are in Table 5.

Again, for both metrics, the originators had a higher centrality than nonoriginators; thus we reject the null hypothesis $H_{2(0)}$.

One may also notice that the difference in both betweenness and degree is not as drastic as with approvers in Table 4. This result also aligns with our motivation for this hypothesis: originators are central because they are connected to approvers, who are also central.

Table 5: Originator centralities.

Metric	Nonoriginator mean	Originator mean	MWW P-value < .01?	Power
Degree	2.7	10.9	Yes	1.0
Betweenness	1.2	49.7	Yes	1.0

5.4. Developer Networks and Extended Developer Networks. Although we concluded in Section 5.1 that annotating the issue tracking system provides valuable information, this situation is not always feasible. Issue tracking annotations require either participation by the development team or manual inspection post hoc (as in this study). But, the regular developer network may *resemble* the extended network enough to still be useful.

One resemblance between the two networks could be the relative developer centralities. That is, are the central developers of the developer network central to the extended developer network? Thus, we test the following hypothesis.

H_3: *Central developers in regular developer networks are also central in extended developer networks.*

The null version of this hypothesis is as follows.

$H_{3(0)}$: *Central developers in regular developer networks are not necessarily central in extended developer networks.*

For this analysis, we use the nonparametric Spearman rank correlation coefficient between the developer centralities. We use a statistical test based on rank because the scales of developer centrality differ. For example, a degree of 10 may be considered high in one network and low in another. Centralities of developers that were only in the extended developer network were excluded from this test. We report the correlation coefficient, along with its statistical significance, in Table 6.

These correlations are fairly strong, indicating that developer centrality of an extended network is good estimators of the developer centrality of an extended network. Thus, we reject hypothesis $H_{3(0)}$.

With these correlations, we investigated further into whether or not the developer centrality from a regular developer network is correlated with being an approver or not. We used the Mann-Whitney-Wilcoxon test again to see if there are any differences in the centrality of the regular developer network between approvers and nonapprovers. Our results for approvers can be found in Table 7.

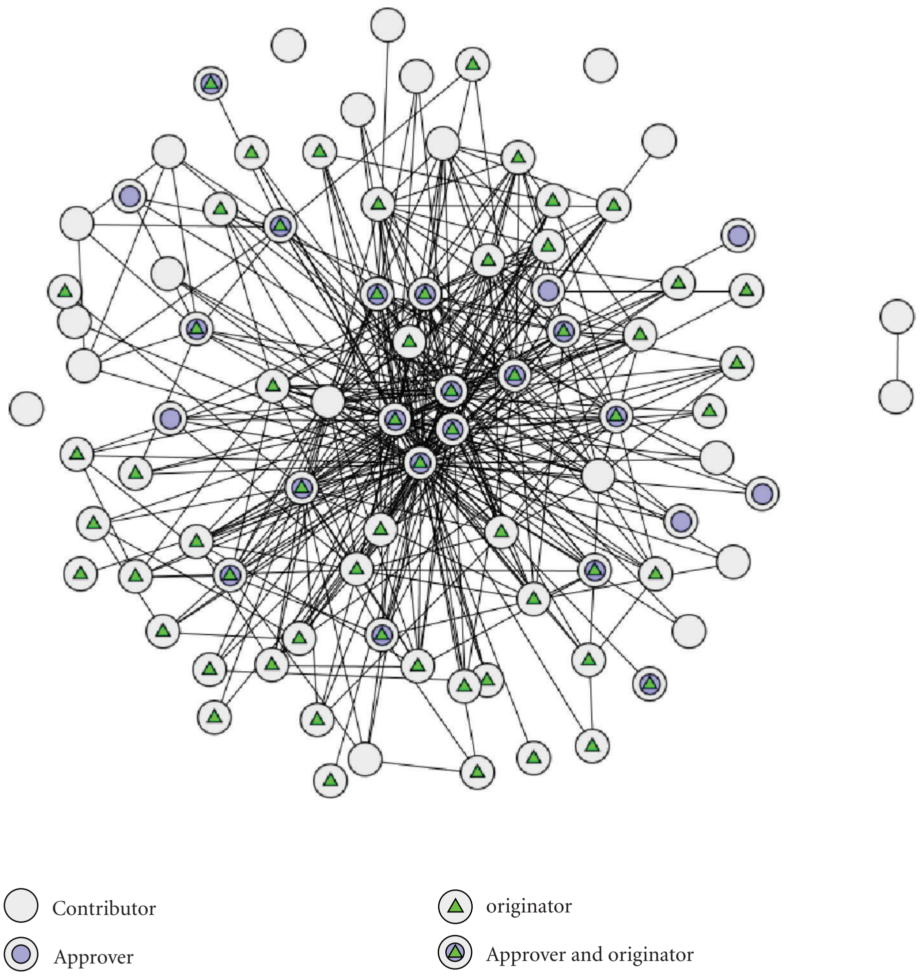


Figure 3: Extended developer network with annotations.

Table 6: Correlation between centralities of regular and extended developer network.

Metric	Spearman between regular and rxtended	P-value < .01?
Degree	0.67	Yes
Betweenness	0.85	Yes

Table 7: Approver centralities, regular developer network.

Metric	Nonapprover mean	Approver mean	MWW P-value < .01?	Power
Degree	1.9	9.5	Yes	1.0
Betweenness	0.3	29.2	Yes	1.0

For both metrics, developer centrality from the regular developer network was higher for approvers than for nonapprovers.

We also investigated whether or not central developers in the regular developer network are also originators. Our results can be found in Table 8.

Table 8: Originator centralities from regular developer network.

Metric	Nonoriginator mean	Originator mean	MWW P-value < .01?	Power
Degree	2.15	6.2	No	1.0
Betweenness	0.5	17.3	Yes	1.0

While not enough empirical evidence exists to say that the Degree measurements were different for originators, there is enough statistical evidence to say that the Betweenness was different for originators.

As with our results in Sections 5.2 and 5.3, the differences in centralities were not as great for the originators as the differences for the approvers. This result provides more evidence that approvers are the most central developers, and originators are central because they are connected to approvers.

5.5. Are Approvers in More Clusters? Expert experience [11] and empirical studies [4–6] have shown that open source communities (e.g., Linux (http://www.linux.org/), Apache

Table 9: Cluster ranks of approvers and nonapprovers from extended developer networks.

	Approvers	Nonapprovers	MWW P-value < .01?	Power
Mean cluster rank	28.3	42.1	Yes	0.86

(http://httpd.apache.org/), and PosgreSQL (http://www.postgresql.org/)) tend to self-organize into smaller subcommunities in large development efforts. However, with many small subcommunities, developers must also coordinate all of those development efforts.

One of the techniques for detecting communities in social networks is the Girvan-Newman hierarchical clustering algorithm (described in Section 2). The output of the Girvan-Newman algorithm is a hierarchy of developers according to the number and size of the clusters they belong to.

We hypothesize that if a developer has enough authority in the project to be a solution approver, then that developer also belongs to more clusters. Thus, we evaluate the following hypothesis.

H_4: *Approvers belong to more developer subcommunities than nonapprovers.*

The null version of this hypothesis is as follows.

$H_{4(0)}$: *Approvers belong to as many developer subcommunities as nonapprovers.*

We perform our evaluation by running the Girvan-Newman hierarchical clustering algorithm on both our annotated developer network and regular developer network. For each developer in the network, we assign an integer rank (which we call *cluster rank*) according to the output of the clustering algorithm. A rank of one indicates that the developer is in more clusters than any other developer.

To evaluate whether being approver is statistically associated with having a high cluster rank, we used the Mann-Whitney-Wilcoxon test between approvers and nonapprovers in the extended developer network. We also report the mean rank of each population. Our results can be found in Table 9.

The difference in cluster ranks between approvers and nonapprovers was statistically significant. Therefore, we reject the null hypothesis $H_{4(0)}$ that approvers are as likely to be in multiple developer subcommunities than nonapprovers.

Interestingly, the lowest rank we observed for an approver was 56. In that situation, the approver only approved one ticket that was closely related to her own work. Other low-ranking approvers were in similar situations where the authority for approving a ticket was (implicitly or explicitly) given to them based on the scope of the work itself. These results indicate that the authority to approve a few tickets may not be related to the overall authority of the project.

6. Discussion

In this section, we discuss some of the ramifications of our results, including relation to prior work and to future work.

6.1. Are Regular Developer Networks Invalidated? Our results from Tables 2 and 3 in Section 5.1 show that the version control change logs do not reflect the entire OpenMRS developer community, and that the issue tracking system does contain information about more developers. These results, however, do not contradict nor invalidate previous studies relying only on version control change logs. Our findings in Section 5.4 show that regular developer networks closely resemble the extended developer network in terms of developer centrality.

Instead, the results of this paper shed more light on who central developers are in open source projects: they tend to be people who are contributing solutions to code that other people are also working on. In the past, experts [11] have observed this concept from experience. While open source communities might be perceived as crowds of people all coding at once, most open source communities have a core group of developers who must guide the direction of the project one issue ticket at a time.

6.2. Why Not Include All Comments? We do not recommend applying all of the comment entries from the issue tracking system to the developer network. In examining the issue tracking system, we found many unrelated comments in the discussion. For example, sometimes people would comment on a ticket with similar (but not identical) issue. That issue would be forked into a new ticket and is unrelated to the original ticket. Or, some people would not provide helpful, constructive comments that would lead to a tangible contribution to the issue. In our analysis, we specifically chose the originator and approver annotations as noise-reducing mechanisms. Using many e-mail archives or other collaboration artifacts also has this kind of problem.

The process of annotating a historical record, however, is manual and can be labor-intensive. One approach is to apply text mining or other machine learning techniques to automatically identify approvers and originators on a ticket; however such a technique would also be noisy. We believe that the open source community should employ a convention of attributing the originator and approver of a given solution to a ticket.

The Linux kernel (http://git.kernel.org/) is one such community that curates its own data with respect to originators and approvers. Their version control system separates the concepts of “committer” and “author” for a given source code change, which was the inspiration for our “originator” annotation. Furthermore, the Linux kernel community has the convention of having a “Signed-off by” field in their version control commit comments that closely resembles our notion of “approver”.

7. Limitations

The scope of this study is only on the technical practice of providing coded solutions (via version control or patches). Many other non-developers exist in open source projects that are not being captured by the development artifacts we are studying.

Additionally, our analysis only includes one case study of an open source project. We chose the OpenMRS project as a production-level system with an active community; so we believe that OpenMRS is a good representative of the open source development community. Our techniques would need to be applied to other software development projects to achieve more generality. Also, the manual process of annotating issues could introduce errors in our data set. To mitigate this factor, we had both researchers perform the annotation separately and then resolve any differences. Lastly, the results of this study are correlations; so we cannot claim any causal direction. For example, we do not know if being a central developer causes being an approver, or vice versa.

8. Conclusion

The objective of this research is to improve the information gained by measurements of developer collaboration by introducing and analyzing two issue tracking annotations: solution originator and solution approver. We found that applying issue tracking annotations revealed more developers than found in the version control logs. The additional results of our study are as follows.

(i) Approvers have a higher developer network centrality than nonapprovers.

(ii) Originators have a higher developer network centrality than nonoriginators.

(iii) Developer networks without issue tracking annotations resemble developer networks with issue tracking annotations in terms of developer centrality.

(iv) Approvers belong to more developer subcommunities than nonapprovers.

While using annotations captures more information about developer collaboration, we found that regular developer networks are still accurate representations of the development community. This result indicates that a developer network can be used as an estimate for developer collaboration and is, therefore, useful in situations where annotations are not available. Our results are an improvement in developer activity metrics that strengthens the connection between what we can measure in the project development artifacts and the team's collaborative problem-solving process.

Appendix

See Figures 2 and 3.

Acknowledgments

The authors thank Mackenzie Corcoran for her contributions to the data collection. This research is supported by the Army Research Office managed by the North Carolina State University Secure Open System Initiative (SOSI). They also thank the OpenMRS development community for opening their data sets to be analyzed.

References

[1] A. Meneely and L. Williams, "Secure open source collaboration: an empirical study of Linus' law," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pp. 453–462, Chicago, Ill, USA, November 2009.

[2] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting failures with developer networks and social network analysis," in *Proceedings of the 16th ACM International Symposium on the Foundations of Software Engineering (SIGSOFT '08)*, pp. 13–23, Atlanta, Ga, USA, November 2008.

[3] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures?" in *Proceedings of the 16th ACM International Symposium on the Foundations of Software Engineering (SIGSOFT '08)*, pp. 2–12, Atlanta, Ga, USA, November 2008.

[4] C. Bird, A. Gourley, P. Devanbu et al., "Mining email social networks in postgres," in *Proceedings of the 3rd International Workshop on Mining Software Repositories (MSR '06)*, pp. 185–186, Shanghai, China, 2006.

[5] 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 International Symposium on the Foundations of Software Engineering (SIGSOFT '08)*, pp. 24–35, Atlanta, Ga, USA, November 2008.

[6] J. M. Gonzales-Barahona, L. Lopez-Fernandez, and G. Robles, "Applying social network analysis to the information in CVS repositories," in *Proceedings of the International Workshop on Mining Software Repositories (MSR '05)*, pp. 1–8, Edinburgh, UK, 2005.

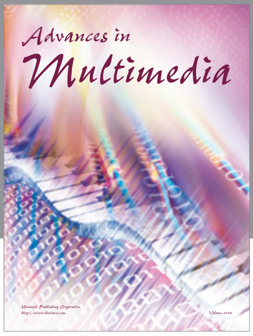
[7] A. Meneely, M. Corcoran, and L. Williams, "Improving developer activity metrics with issue tracking annotations," in *Proceedings of the Workshop on Emerging Trends in Software Metrics (WETSoM '10)*, pp. 75–80, Cape Town, South Africa, 2010.

[8] U. Brandes and T. Erlebach, *Network Analysis: Methodological Foundations*, Springer, Berlin, Germany, 2005.

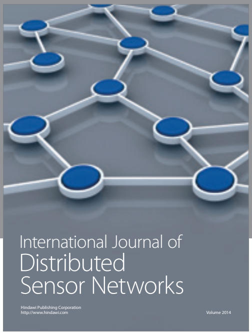
[9] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: interactive visual exploration of socio-technical relationships in software development," in *Proceedings of the 31st International Conference on Software Engineering (ICSE '09)*, pp. 23–33, May 2009.

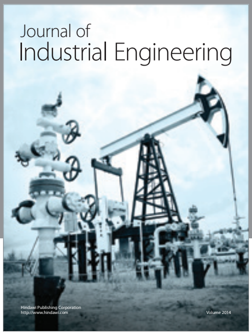
[10] N. Nagappan, B. Murphy, and V. R. Basili, "The influence of organizational structure on software quality: an empirical case study," in *Proceedings of the 30th International Conference on Software Engineering (ICSE '08)*, pp. 521–530, Leipzig, Germany, May 2008.

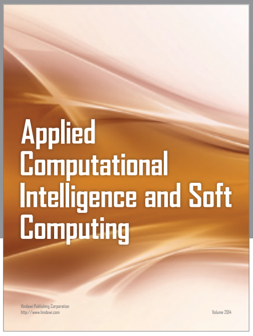
[11] E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly and Associates, Sebastopol, Calif, USA, 1999.

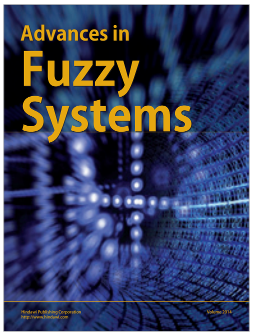

Advances in Multimedia

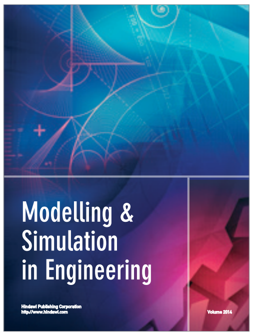
The Scientific World Journal

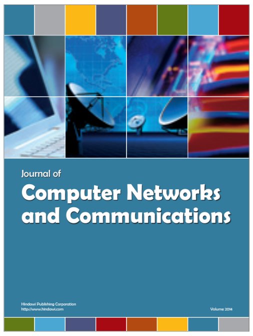

International Journal of Distributed Sensor Networks

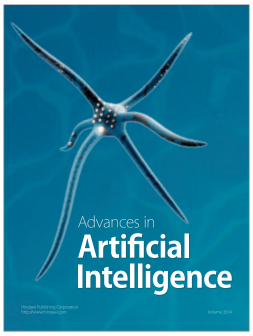

Journal of Industrial Engineering

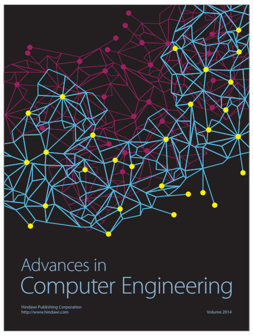

Applied Computational Intelligence and Soft Computing


Advances in Fuzzy Systems

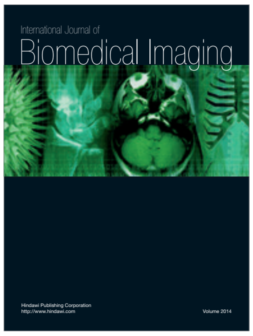

Modelling & Simulation in Engineering

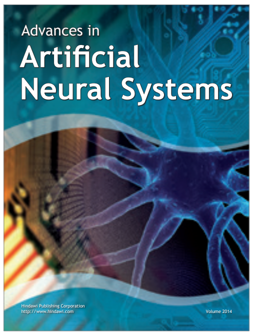

Journal of Computer Networks and Communications

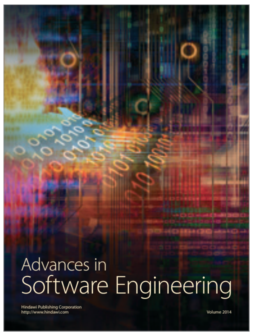

Advances in Artificial Intelligence


Advances in Computer Engineering

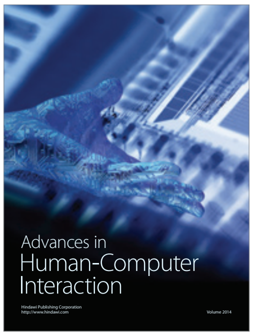
International Journal of Computer Games Technology

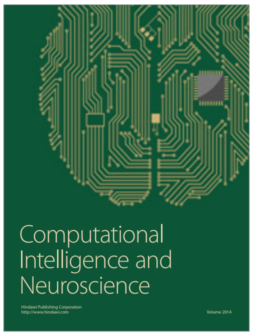

International Journal of Biomedical Imaging


Advances in Artificial Neural Systems


Advances in Software Engineering

Journal of Robotics


Advances in Human-Computer Interaction


Computational Intelligence and Neuroscience

International Journal of Reconfigurable Computing

Journal of Electrical and Computer Engineering