Improving Developer Activity Metrics with Issue Tracking Annotations

Andrew Meneely, Mackenzie Corcoran, Laurie Williams
North Carolina State University
{apmeneel, mhcorcor, lawilli3}@ncsu.edu

ABSTRACT
Understanding and measuring how groups of developers collaborate on software projects can provide valuable insight into software quality and the software development process. Current practices of measuring developer collaboration (e.g. with social network analysis) usually employ metrics based on version control change log data to determine who is working on which part of the system. Version control change logs, however, do not tell the whole story. Information about the collaborative problem-solving process is also documented in the issue tracking systems that record solutions to failures, feature requests, or other development tasks. To enrich the data gained from version control change logs, we propose two annotations to be used in issue tracking systems: solution originator and solution approver. We examined the online discussions of 602 issues from the OpenMRS healthcare web application, annotating which developers were the originators of the solution to the issue, or were the approvers of the solution. We used these annotations to augment the version control change logs and found 47 more contributors to the OpenMRS project than the original 40 found in the version control change logs. Applying social network analysis to the data, we found that central developers in a developer network have a high likelihood of being approvers. These results indicate that using our two issue tracking annotations identify project collaborators that version control change logs miss. However, in the absence of our annotations, developer network centrality can be used as an estimate of the project's solution approvers. This improvement in developer activity metrics provides a valuable connection between what we can measure in the project development artifacts and the team's problem-solving process.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *process metrics, product metrics*

General Terms
Measurement, Human Factors

Keywords
Developer, collaboration, metric, network analysis

WETSoM'10, May 4, 2010, Cape Town, South Africa.

1. INTRODUCTION
Understanding and measuring how groups of developers collaborate on software projects can provide valuable insight into software quality and the software development process. One class of metrics, called *developer activity metrics*, analyze the structure of development in terms of how developers collaborate [5]. Already, several applications have been found for developer activity metrics, including predicting failures [6, 8], predicting vulnerabilities [5], and studying individual projects [1, 2, 4].

Many of these developer activity metrics employ data obtained from 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[1]. 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 by two issue tracking annotations: solution originator and solution approver.* We examined the

[1] Taken from http://dev.openmrs.org/ticket/1171

online discussions of 602 tickets from the OpenMRS[2] 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.

The rest of the paper is organized as follows. Section 2 describes background related to developer activity metrics, centrality, developer networks, and issue tracking systems. Section 3 describes our data collection process. Section 4 describes our analysis and results. Section 5 summarizes our conclusions.

2. BACKGROUND

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 [3]. In network analysis, vertices of a graph are called nodes, and edges are called connections. A sequence of non-repeating, 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 [3] of node *n* is defined as the number of geodesic paths that include *n*. A high betweenness means a high centrality.

The specific network we are examining in this paper is known as a **developer network** [2, 5, 6]. Developers are represented as nodes, and edges exist between two nodes where two developers made contributions to the same source code file around the same period of time (we use one month on this study). The result is an undirected, unweighted, and simple graph where each node represents a developer and edges are based on whether or not they have worked on the same file within a specified period of time. For example, suppose we have the contribution table found in Table 1.

[2] http://openmrs.org

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.

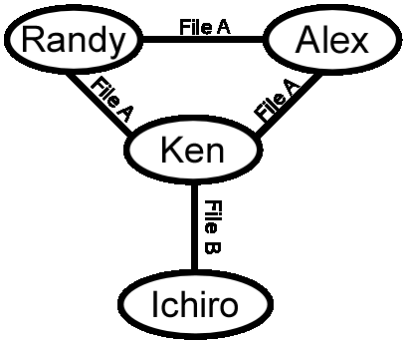

Figure 1: Resulting developer network from the Table 1

In the above network, 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 [2, 5, 6, 10].

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.

3. RELATED WORK

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 and Lopez-Fernandez [4] 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. [2] 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 sub-communities 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 sub-communities 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. [8] 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., [6] 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. [10] 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. [7] 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. Among the findings is that more edits made by many, non-cohesive developers leads 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. [8] in that both models attempt to differentiate healthy changes in software from the problematic changes.

Lastly, Meneely and Williams [5] applied developer activity metrics to security data in examining aspects of the saying "Many eyes make all bugs shallow" (known as Linus' Law [9]). Using both developer networks and the contribution networks proposed by Pinzger et al. [8], 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.

4. EXTENDING DEVELOPER NETWORKS

In prior research [2, 4-8], 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).

In this study, we extend the notion of contributions beyond version control change logs to originators and approvers of solutions in issue tracking systems. 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 5. 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[3] or Trac[4].

5. COLLECTING ANNOTATIONS

The developers of the OpenMRS project use Trac for their issue tracking system and Subversion[5] (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.

[3] http://www.bugzilla.org

[4] http://trac.edgewall.org

[5] http://subversion.tigris.org

We considered a person to be an originator if:

- A person submitted a patch that was a major part of the solution to the issue;
- A person introduced code (e.g. wrote code directly into the ticket's message board), and that code was used in the solution; or
- The committer of the solution attributed someone on the ticket in their Subversion commit message.

We considered a person to be an approver if:

- The person changed the ticket status to "approved";
- 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
- 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, 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.

6. EVALUATION

In this section, we examine whether information about developer collaboration can be gained by analyzing by two issue tracking annotations: solution originator and solution approver.

6.1 How much information is gained?

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 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 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%).

Table 2: Approvers in regular and extended developer networks

	Only in Regular DN	Only in Extended DN	Total
Non-Approver	24 (60%)	39 (82%)	63 (72%)
Approver	16 (40%)	8 (18%)	24 (27%)
Total	**40 (100%)**	**47 (100%)**	**87 (100%)**

The high rate of non-approvers 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).

Table 3: Originators in regular and extended developer networks.

	Only Regular DN	Only in Extended DN	Total
Non-Originator	13 (32%)	7 (15%)	20 (23%)
Originator	27 (68%)	40 (85%)	67 (77%)
Total	**40 (100%)**	**47 (100%)**	**87 (100%)**

Additionally, the complexity of the network structure increased dramatically. Figure 2 (shown in 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 non-approvers are being accounted for.

6.2 Are Approvers Central?

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 [6], we found developer centrality can be used for predicting failures in files.

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 non-approvers.*

In this paper, we use two measures of centrality: degree and betweenness (defined in Section 2). We used the non-parametric Mann-Whitney-Wilcoxon test ($p<0.05$) to examine the differences in centrality between approvers and non-approvers. Since we are evaluating multiple hypotheses, we used a Bonferroni correction, so we check our p-values against 0.01 instead of the traditional 0.05. Table 4 shows our results:

Table 4: Approver centralities

Metric	Non-Approver Mean	Approver Mean	MWW p-value <0.01?
Degree	6.0	16.9	Yes
Betweenness	2.7	132.6	Yes

For both metrics, the approvers had a higher developer centrality than non-approvers, thus we accept hypothesis H_1. Therefore, central developers of the extended developer network are also approving solutions to the issue tickets. Figure 3 (shown in Appendix) visually illustrates that the approvers are generally well-connected developers in the network.

6.3 Are Originators Central?

Centrality is related not only to *how many* connections you have, but *to whom* you are connected (e.g. if you are connected to a very central developer, you become central yourself). Since originators and approvers on the same ticket are automatically connected to each other, it follows that 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 non-originators.*

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

Table 5: Originator centralities

Metric	Non-Originator Mean	Originator Mean	MWW p-value <0.01?
Degree	2.7	10.9	Yes
Betweenness	1.2	49.7	Yes

Again, for both metrics, the originators had a higher centrality than non-originators, thus we accept H_2.

One may also notice that the difference in both betweenness and degree is not as drastic as with approvers in Table 5. This result also aligns with our motivation for this hypothesis: originators are central because they are connected to approvers, who are also central.

6.4 Do Developer Networks Resemble Extended Developer Networks?

Although we concluded in Section 6.1 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.*

For this analysis, we use the non-parametric 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.

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

Metric	Spearman between Regular and Extended	p-value <0.01?
Degree	0.67	Yes
Betweenness	0.85	Yes

These correlations are fairly strong, indicating that developer centrality of an extended network are good estimators of the developer centrality of an extended network. Thus, we accept hypothesis H_3.

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 non-approvers. Our results for approvers can be found in Table 7.

Table 7: Approver centralities, regular developer network

Metric	Non-Approver Mean	Approver Mean	MWW p-value <0.01?
Degree	1.9	9.5	Yes
Betweenness	0.3	29.2	Yes

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

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	Non-Originator Mean	Originator Mean	MWW p-value <0.01?
Degree	2.15	6.2	No
Betweenness	0.5	17.3	Yes

While there is not enough evidence 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 Section 6.2 and 6.3, 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.

These results indicate that one can use developer centrality of a developer network as an estimate of a developer being an approver or originator, with the exception of Degree for originators. This result is particularly helpful in situations where issue tracking annotations, (or even any issue tracking data) is not available or feasible collect post hoc.

7. CONCLUSION

The objective of this research is to improve the information gained by measurements of developer collaboration by introducing and analyzing by two issue tracking annotations: solution originator and solution approver. We examined the value of these annotations in measuring developer collaboration. We found that applying the solution originator annotation introduced many new contributors not revealed by the version control change logs. Furthermore, we found that both originators and approvers are central developers to the extended developer network. Lastly, while using annotations appear to capture more information about developer collaboration, we found that the developer centralities of a developer network are correlated with the developer centralities of the extended developer network. 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. This improvement in developer activity metrics provides a valuable connection between what we can measure in the project development artifacts and the team's problem-solving process.

8. ACKNOWLEDGMENTS

This research is supported by the Army Research Office managed by the North Carolina State University Secure Open System Initiative (SOSI). We also thank the OpenMRS development community for opening their data sets to be analyzed.

9. REFERENCES

[1] C. Bird, A. Gourley, P. Devanbu *et al.*, "Mining Email Social Networks in Postgres," in Mining Software Repositories, Shanghai, China, 2006, p. 185-186.

[2] C. Bird, D. Pattison, R. D'Souza *et al.*, "Latent Social Structures in Open Source Projects," in FSE, Atlanta, GA, 2008, p. p24-36.

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

[4] J. M. Gonzales-Barahona, L. Lopez-Fernandez, and G. Robles, "Applying Social Network Analysis to the Information in CVS Repositories," in 2005 Mining Software Repositories, Edinburgh, Scotland, United Kingdom, 2004, p.

[5] A. Meneely, and L. Williams, "Secure Open Source Collaboration: An Empirical Study of Linus' Law " in Computer and Communications Security, Chicago, IL, 2009, p. 453-462.

[6] A. Meneely, L. Williams, J. Osborne *et al.*, "Predicting Failures with Developer Networks and Social Network Analysis " in Foundations in Software Engineering, Atlanta, GA, 2008, p. 13-23.

[7] N. Nagappan, B. Murphy, and V. R. Basili, "The Influence of Organizational Structure on Software Quality," in International Conference on Software Engineering, Leipzig, Germany, 2008, p. 521-530.

[8] M. Pinzger, N. Nagappan, and B. Murphy, "Can Developer-Module Networks Predict Failures?," in Foundations in Software Engineering, Atlanta, GA, 2008, p. 2-12.

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

[10] A. Sarma, L. Maccherone, P. Wagstrom *et al.*, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, 2009, pp. 22-33.

Appendix

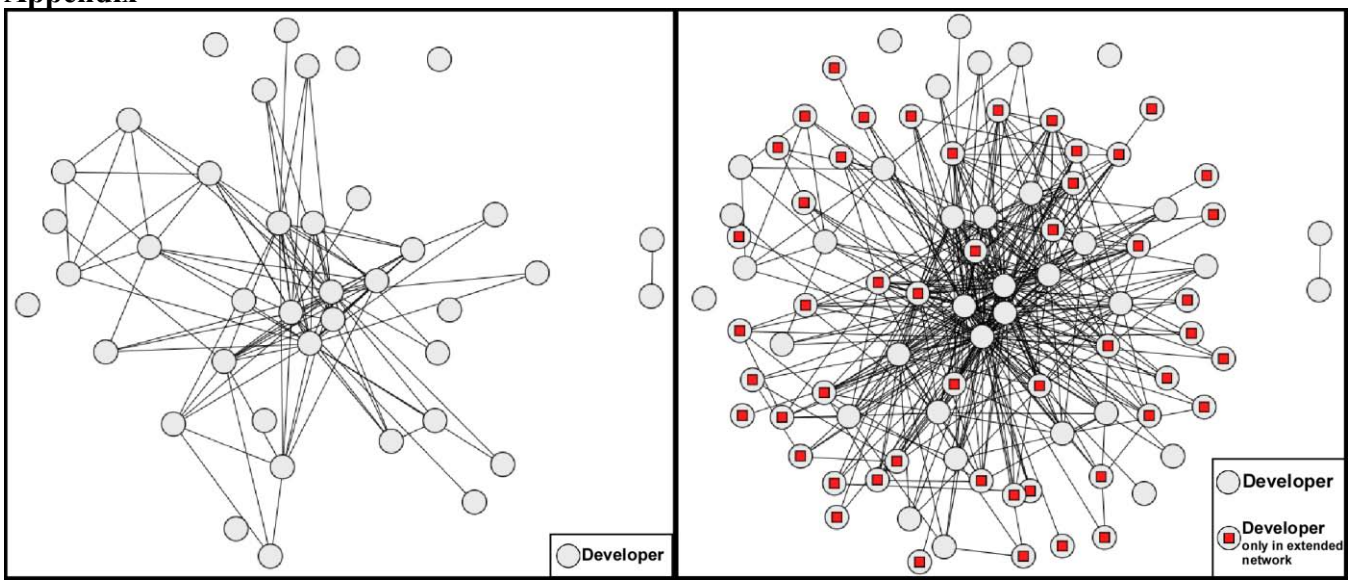

Figure 2: (left) Regular developer network, and (right) extended developer network of the same layout

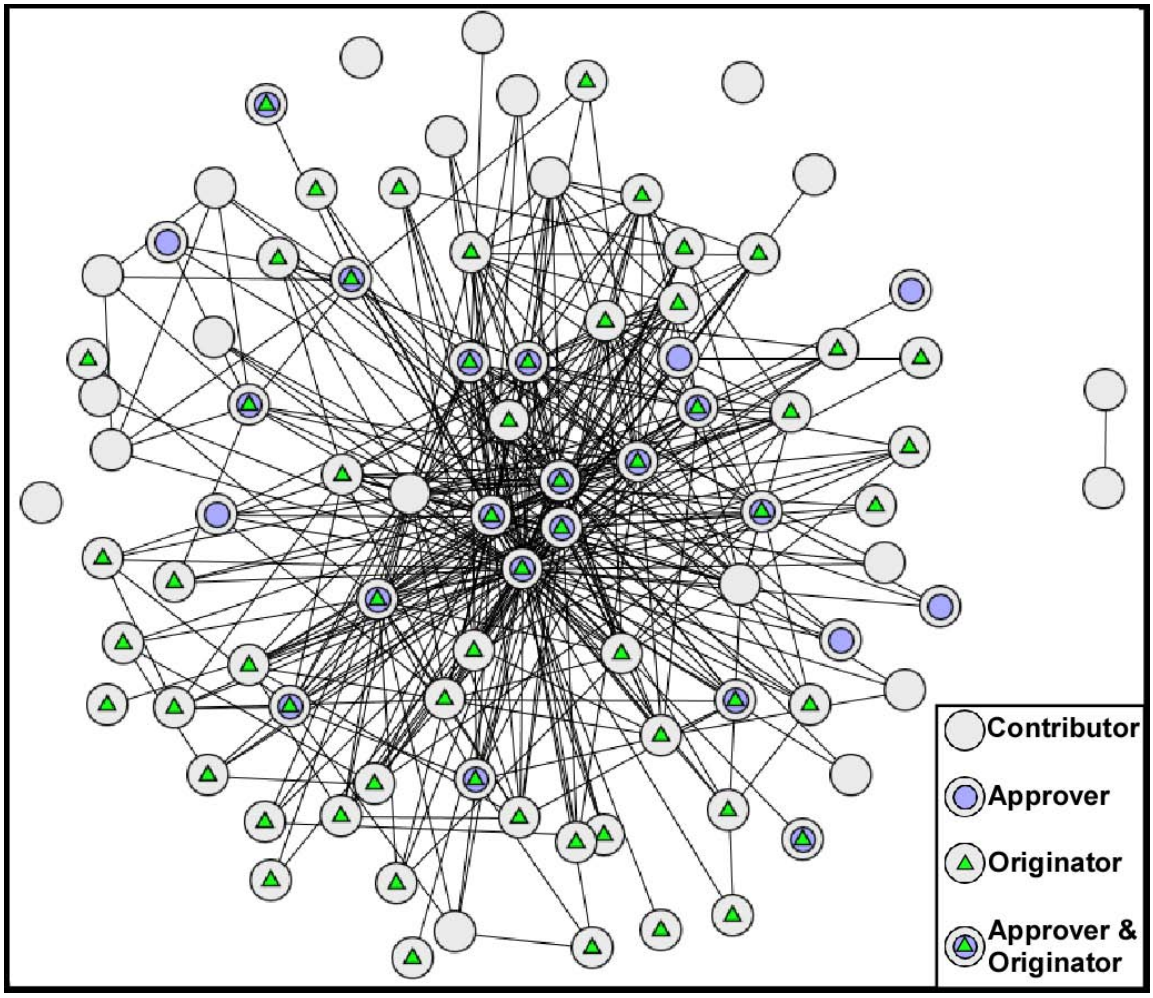

Figure 3: Extended developer network with annotations