Recommending People in Developers' Collaboration Network

Didi Surian*, Nian Liu†, David Lo*, Hanghang Tong‡ Ee-Peng Lim* and Christos Faloutsos§
*School of Information Systems, Singapore Management University
Email: {didisurian, davidlo, eplim}@smu.edu.sg
†College of Software, Zhejiang University
Email: squall405001206@126.com
‡IBM T.J. Watson Research, 19 Skyline Dr., Hawthorne NY, 10532
Email: htong@us.ibm.com
§Department of Computer Science, Carnegie Mellon University
Email: christos@cs.cmu.edu

***Abstract*—Many software developments involve collaborations of developers across the globe. This is true for both open-source and closed-source development efforts. Developers collaborate on different projects of various types. As with any other teamwork endeavors, finding compatibility among members in a development team is helpful towards the realization of the team's goal. Compatible members tend to share similar programming style and naming strategy, communicate well with one another, etc. However, finding the right person to work with is not an easy task. In this work, we extract information available from Sourceforge.Net, the largest database of open source software, and build developer collaboration network comprising of information on developers, projects, and project properties. Based on an input developer, we then recommend a list of top developers that are most compatible based on their programming language skills, past projects and project categories they have worked on before, via a random walk with restart procedure. Our quantitative and qualitative experiments show that we are able to recommend reasonable developer candidates from snapshots of Sourceforge.Net consisting of tens of thousands of developers and projects, and hundreds of project properties.**

I. INTRODUCTION

More and more software projects are developed in a collaborative fashion involving developers across the globe. This is true both for industrial and open source projects. Microsoft has a development center in India, China, USA, etc. Many software sites enable developers to start, join, and participate in different projects. Eclipse, Apache Tomcat, MySQL, etc are examples of successful software projects that come from a globally distributed team. Among these sites, a super-repository Sourceforge.Net contains the largest amount of information on developers working on various projects of various types. A super-repository is an integrated archive of multiple projects and contains many information including the people contributing to various projects, some project properties such as project categories and programming languages that build that projects.

Similar to many teamwork endeavors, having a team that works closely together is essential towards the realization of the team's goal. Not everyone works well with everyone else. Different individuals have different working styles and preferences. The danger of a dysfunctional team is arguably higher in a distributed development setting than among developers working from the same location.

In this work, we would like to help construction of workable teams especially in a distributed setting and extend the scale of collaboration among developers. To achieve this goal, we develop a new approach to recommend people from the implicit developer's social or collaboration network extracted from a database of software projects. Our tool could be used by project managers, or developers in open-source settings to know other potentially compatible developers based on past history of their interactions.

There have been several recent studies on developing a recommendation system to aid software development. Most of these studies develop a recommendation system to aid a software developer in executing a software maintenance task [34], [23]. There is a number of recent work that utilizes developers social network to predict for failures [21], [32]. Most related to our work is the work by Ma *et al.* that develops a developer recommendation system based on usage expertise [16]. While the work by Ma *et al.* focuses on fine-grained file-level history of how developers make changes to different source code files in a system, our work focuses on coarse-grained history on which projects and project types developers work. Different from the work by Ma *et al* which works on two projects, we work on a super-repository, c.f., [15], that consists of 151,776 projects, 354 project categories, and 50 different programming languages. Another related work is by Lungu *et al.* and Sarma *et al.* which proposes approaches to manage a super-repository via visualization strategies [15], [25]. We believe our approach is complementary to that of Lungu *et al.*'s and Sarma *et al.*'s by providing a list of people to be recommended to a particular developer which would complement the visualization of the super-repository.

Our approach starts by extracting a graph from past history of projects and developers working on them. The graph contains three types of nodes: developers, projects, and project

properties. We consider project properties as project category and programming language. An edge is introduced between a developer and a project, if the developer has worked on the project before. An edge is introduced between a project and a property if the project has a particular property (e.g., (written in) C++, Java, (about) Database, etc). With a graph representation, we could handle heterogeneity inherent in software data. We desire a metric to quantify the "social distance" from one developer to another in the developers' network, which takes into account the projects the developers worked on, the properties of the projects they are involved with before, and the transitive relationships between the developers (e.g., a collaborator of a collaborator). We find the similarity score computed by performing random walk with restart (RWR) used in search engines to be appropriate. RWR could capture multi-faceted relationship between two nodes in the graph. This score is then used to obtain a list of compatible people to be recommended to the input developer.

We perform a qualitative experiment on a network of 209,009 developers working on 151,776 projects extracted from Sourceforge.Net[1]. We recommend people for 20 different developers and show that our recommendation is reasonable. We could automatically recommend a sorted list of other developers based on the projects and properties of projects that they have worked before. Developers having more similarity to the input developer would be ranked first in the list. The process is shown to be scalable as it could be completed in a few seconds. We also perform a quantitative experiment on pairs of consecutive snapshots from SourceForge.Net. A recommendation is good if it eventually leads to a collaboration among developers. We evaluate the proportion of the new collaborations that correspond to recommendations made by our tool. The result shows that up to 83.33% accuracy could be attained.

The structure of the paper is as follows. In Section II, we discuss related work. We present our approach in Section III. We describe our experiment results in Section IV. We discuss some issues in Section V. We finally conclude and describe future work in Section VI.

II. Related Work

Recently there has been an active interest to mine knowledge from the mass of software data in various formats. Some studies proposed mining of knowledge from code [10], [11], [28], [18], from execution traces [30], [26], [7], from software repositories like SVN or CVS [34], [20], etc. In many of these studies, a data mining algorithm is deployed to solve a software engineering task. In this work, we are also interested in mining knowledge from the mass of software data. In particular, we focus on developers' collaboration network with the purpose of recommending compatible developers. We make use of direct and transitive history of collaborations (based on information on past projects and project properties such as project categories and project programming languages) to recommend a personalized list of recommended developers. Random walk with restart is used to realize this goal.

There have been several recent work on developing a recommendation system to aid software development. Most of these work focus on helping developers with a software maintenance task. Gall *et al.* and Zimmermann *et al.* proposed such recommendation systems based on change sets i.e., the sets of program elements that change together [8], [34]. Rastkar and Murphy argued that developers' interaction could complement the information from change sets [23]. In this work, we build a recommendation system not to predict future maintenance activities, rather to recommend compatible developers with similar history of past projects, or categories of projects worked before, or programming language skills.

Social network analysis has drawn much research interests in various communities [6], [14], [13], [2], [3], [21]. In the software engineering community, there is a number of recent work that utilizes developers' social network information. Bird *et al.* extracted a social network from developers' email communications [3]. They found that the level of email activities correlates with the amount of changes in the source code. Pinzger *et al.* developed an approach to predict failure based on social networks induced upon dependencies on the code that developers wrote [21]. Wolf *et al.* extended the work to consider social network built upon communication among developers [32]. Surian *et al.* mined for frequent patterns of collaborations [27]. Our work enriches past studies that leverage developers' social network, by proposing a new approach to recommend people based on the projects and project properties they have worked before.

A recent work by Chen *et al.* proposed an approach to recommend friends on social networking sites [5]. Our work also has a similar flavor. However, we make use of specialized information on developers' social/collaboration networks namely projects and project properties to build a customized recommendation system for software developers working in a super-repository. The work by Chen *et al.* focuses on informal friendship-based social network, on the other hand, our work focuses on semi-formal working relationship among developers in a software super-repository.

There is also a recent work by Lappas *et al.* that develops a new algorithm to recommend a team of experts from a social network graph [12]. There are two differences between their work and ours. First, our goal is to extract a list of people relevant to a particular input developer. The work by Lappas *et al.* on the other hand extracts a set of people relevant to a particular task. We believe it is possible to merge the two approaches in the future. Second, different from Lappas *et al.*'s work, we focus on developers' social/collaboration network with information on developers, projects, and project properties such as project categories and project programming languages.

Ma *et al.* developed a developer recommendation system based on the set of files developers have worked on in the past [16]. We extend the work by Ma *et al.* by considering a higher level of abstraction namely the projects and project

[1] http://sourceforge.net

properties the developers have worked on. We also extend the work by Ma *et al.* to analyze a super-repository containing numerous projects rather than a small number of related repositories. We believe considering both lower (i.e., files) and higher (i.e., projects and project properties) levels of abstraction is an interesting future work.

Lungu *et al.* proposed an approach to visualize a super-repository [15]. A related visualization study is also performed by Sarma *et al.* [25]. Wermelinger *et al.* [31] proposed an application of formal concept analysis (FCA) to compute and visualize the hierarchical ordering of socio-technical relations. We believe our approach is complementary to that of Lungu *et al.*'s, Sarma *et al.*'s, and Wermelinger *et al.*'s. By using both visualization and our approach, users could benefit not only from the visualization to obtain a general overview of the super-repository, but also from a more specific information provided by our system, namely the list of developers that are likely to be compatible to an input developer in question.

There have also been several work on the analysis of open source developer communities [24], [33], [17]. Ricca *et al.*[24] proposed an approach to identify *Heroes* in open source projects. *Heroes* refer to developers who manage large and critical portions of a system. Xu *et al.* and Madey *et al.* also studied the open-source software development community at SourceForge. Xu *et al.*'s work is related to community distributions. Madey *et al.* analyzed developer collaboration networks in open-source community by using social network theory. Our work is complementary to that of Xu *et al.*'s and Madey *et al.*'s by developing a recommender tool utilizing the past history of projects and developers working on them.

There is a number of studies analyzing globally distributed software development activities [9], [22], [4]. Herbsleb *et al.* reported various qualitative experiences learned from nine globally-distributed software development projects within Siemens [9]. Ramasubbu and Balan investigated the cost of globally distributed software development efforts. They highlighted that the cost of dispersion has significant effect on productivity and could be mitigated by deploying structured software engineering processes [22]. Cataldo and Herbsleb reported networks formed by analyzing communications between developers in a globally distributed software project. They found that over time there exists a group of developers that become the liaisons between teams and locations [4]. In this study, we complement the above mentioned studies by proposing a developer recommendation tool that could be used to promote globally distributed software development. As a case study, we investigate an open source globally distributed software development portal namely SourceForge.Net.

III. Proposed Approach

In this section, we describe the formation of Developer-Project-Property (DPP) graph from a super-repository. We then describe our proposed metric to measure the compatibility between two developers. Next, we describe how this metric could be computed fast as various input developers are presented to the system.

A. Formation of DPP Graph

A super-repository usually comes with a set of tables storing information on the various projects, types, developers, etc that are part of the super-repository. Based on these tables, we extract a Developer-Project-Property (DPP) graph as a target representation of the developers' collaboration network.

A DPP graph has a set of nodes and edges. There are three types of nodes: developer, project, and project properties. Project properties correspond to project categories and project programming languages that further describe the project. There are two types of edges: one connecting developer and project (d-p edges), another connecting project and project properties (p-prop) edges. A developer could work on multiple projects. On the other hand, a project could be associated to multiple categories. A project could also have several programming languages used to develop that project. The resultant graph hence is a tripartite graph. An example of such a graph is shown in Figure 1.

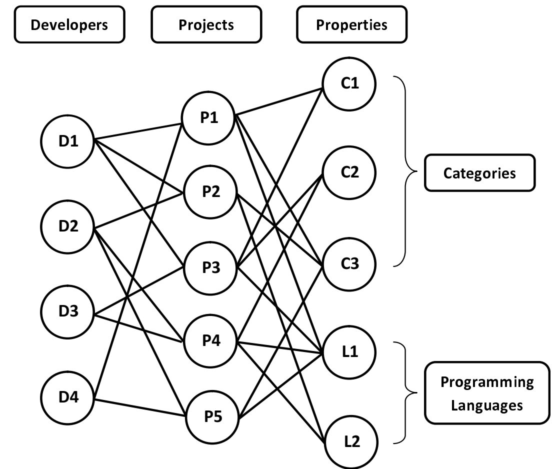

Fig. 1. **Developer-Project-Property (DPP) Graph**

We extract information for the construction of the DPP graph by performing SQL queries to the database containing information on the super-repository. In the case of SourceForge.Net that we study in this paper, we process the raw database described in [1].

B. Compatibility Metric

Next we describe our proposed compatibility metric. More compatible developers should have a higher score than less compatible ones.

Developers in a super-repository worked on various projects. Two developers worked together before on various projects are likely to be more compatible than two developers that have not worked before.

As many developers have not worked with many other developers before, we also make use of project properties. We use project properties such as project categories and programming languages used in the project. Each project is associated

to a list of categories. For example, in SourceForge.Net, the project Notepad++ is associated with the following project categories: Text Editors, Software Development, etc. Developers that have developed similar applications (i.e., applications of the same category but different name) is likely to be more compatible than those developing totally different types of applications. Also, developers with many mutual "colleagues" developing the same/similar projects before is likely to be more compatible than complete "strangers". The same is likely to be true for "colleagues of colleagues" albeit with a lower likelihood. Furthermore, developers also tend to seek partners that have similar programming language skills.

The above describes three qualitative criteria for a good compatibility metric. We find similarity based on random walk with restart (RWR) first proposed in [19] for search engines, to be a suitable metric realizing the above criteria. Given a graph and a node n in the graph, by performing a random walk starting with n, one will reach various other nodes. Nodes that are closer to n are more likely to be visited. Nodes that are far from n are less likely to be visited. One would assign scores to other nodes based on the likelihood the nodes are visited by the random walk with restart starting at n.

Considering a DPP and an input developer node dn in the DPP, developers that have a high score based on RWR on the DPP would have the desired characteristics of likely compatible people to be recommended. They would either have worked on some of the projects dn has worked before, and/or have worked on many similar projects of the same category, and/or share many mutual "colleagues", and/or have similar programming language skills.

C. Fast Computation of the Metric

To compute the compatibility metric, one could directly apply a straightforward implementation of random walk with restart. However, the process would not scale for large graphs: it either requires quadratic space, cubic pre-computa- tion time, or slow response time for a given input developer node in the DPP graph. In order to speed up the process, we employ the approach proposed in [29], namely B_LIN. The approach is briefly described below.

Let W be the normalized adjacency matrix of the graph and $(1-c)$ be the re-start probability of random walk with restart. If we define an $n \times n$ matrix $Q = (I - cW)^{-1}$, where I is an identity matrix. It turns out the ith column of the matrix Q is proportional to the ranking vector of the ith query node. The ranking vector of the ith query node corresponds to the RWR score of the ith node to the other nodes. It is prohibitive to pre-compute and store the full Q matrix in the training stage. On the other hand, simply performing on-the-fly computation of RWR score in the query stage without any pre-computation would require a lot of matrix-vector multiplication, which leads to slow response.

B_LIN carefully balances the pre-computation cost in the training stage and the on-line query cost. The key operation in the algorithm is a so-called block-linear decomposition on the normalized adjacency matrix W. That is, we will approximate W by a block-diagonal matrix W_1 plus a low-rank approximation USV:

$$W \approx W_1 + USV \tag{1}$$

where W_1 is a block-diagonal matrix and it corresponds to the within community (i.e., dense neighborhood of nodes and edges in a graph) links; U, S and V are three low rank matrices and they correspond to the cross-community links.

Then, we have the following approximation for the Q matrix:

$$\begin{aligned} Q &= (I - cW)^{-1} \\ &\approx (I - cW_1 - cUSV)^{-1} \\ &= Q_1 + cQ_1 U \Lambda V Q_1 \end{aligned} \tag{2}$$

where $Q_1 = (I - cW_1)^{-1}$ is easy to compute since we can do inverse for each diagonal block independently, and $\Lambda = (S^{-1} - cVQ_1U)^{-1}$ is another small matrix inverse. In in the training stage, we pre-compute and store Q_1, Λ, U and V, all of which are cheap to compute.

In the on-line query stage, we can get the ith column of Q matrix q_i which contains the RWR scores for q_i by a few matrix-vector multiplications:

$$\begin{aligned} &q_0 \leftarrow Q_1 \; e_1 \\ &q_i \leftarrow V \; q_0 \\ &q_i \leftarrow \Lambda \; q_i \\ &q_i \leftarrow U \; q_i \\ &q_i \leftarrow Q_1 \; q_i \\ &q_i \leftarrow q_0 + cq_i \end{aligned}$$

where e_i is an $n \times 1$ vector with i^{th} element to be 1 and 0s for all the others.

D. Recommending Developers

As the final step, we need to recommend developers based on the RWR compatibility metric. We first train B_LIN on the DPP graph. Next we query B_LIN on the input developer in question. B_LIN computes scores between the input developer to other k nodes in the graph with the input developer. We only keep nodes that are developers and sort them based on the score. A list containing a maximum of top-k developers is then presented to the user as a recommendation to the input developer in question. For each of the recommended developers, we also output the common projects and types they share with the input developer.

The end-to-end pseudocode of our approach is shown in Figure 2. The pseudocode on the left describes the training phase. At line 1, the Developer-Project-Property (DPP) graph is constructed as described in Section III-A. This DPP is later broken down into parts via a graph partitioning algorithm as described in [29] (line 2). Various inverse matrices are pre-computed based on these partitions (line 3). These pre-computed inverse matrices are used to speed up the query phase. The DPP graph and inverse matrices are then output

```
Procedure RecommendDevelopers_Train
Inputs:
  DB: A database representing a super-repository;
Outputs:
  DPP graph representation of DB;
  Pre-computed inverse matrices;
Method:
1:  Let DPP = Create a Developer-Project-Property graph
              from DB
2:  Let DPP_PARTS = Break down the DPP graph
                    into parts following [29]
3:  Let PRECOMPUTE = Pre-compute inverse
                     matrices for graphs in
                     DPP_PARTS
                     following [29]
4:  Output DPP
5:  Output PRECOMPUTE
```

```
Procedure RecommendDevelopers_Query
Inputs:
  DPP: Developer-Project-Property graph;
  devQ: Input developer in question;
  PRECOMPUTE: Pre-computed inverse matrices;
  k: Max. num. of top compatible developers to return;
Outputs:
  A sorted list of recommended developers;
Method:
6:  Compute the RWR score from devQ to other nodes in
      DPP based on PRECOMPUTE following [29]
7:  Let RECOMMEND = Extract top-k developers sorted
                    based on their RWR scores
8:  For each developer devK in RECOMMEND
9:     Let CProj = Find common projects between devQ & devK
10:    Let CCat = Find common project categories btw. devQ & devK
11:    Let CLang = Find common programming lang. btw. devQ & devK
12:    Output (devK,CProj,CCat,CLang)
```

Fig. 2. Developer Recommendation Algorithm: Training and Query Phase

and are used in the query phase (lines 4-5). The pseudocode on the right describes the query phase where given an input developer $devQ$, a list of top compatible developers are returned. First, at line 6, the RWR scores between $devQ$ to the nodes in the DPP graph are computed. The pre-computed inverse matrices $PRECOMPUTE$ makes this process faster. Based on these RWR scores, at line 7, we sort the result in descending order and extract the top-k developers. Information on common projects, project categories, and project programming languages between $devQ$ and top-k developers are computed and output at lines 8-12.

E. Overall Process

The block diagram of our approach is illustrated in Figure 3. Our recommendation tool has two phases: training and query. We would first construct a Developer-Project-Property (DPP) graph from a super-repository. This graph is later pre-processed to form pre-computed matrices in the training phase. The pre-computed matrices would then be used for the query phase. In the query phase, our tool accepts a set of user inputs including the developer desiring recommendation. Our query processor would then produce a list of recommendations and present the top-k most recommended developers. Figure 2 (left) describes the pseudo-code for the pre-processor. Figure 2 (right) describes the pseudo-code for the query-processor.

IV. Experiments

We have conducted both qualitative and quantitative experiments to evaluate our approach. We describe the two experiments in the following sections. We also describe some threats of validity.

A. Qualitative Experiment

In this experiment, we show the result of performing various queries on SourceForge.Net dataset using our recommender tool. We first describe our experiment settings followed with our results.

1) Experiment Settings: We analyze SourceForge.Net, the largest open source software development portal. We use the database dumps of SourceForge.Net collected by Madey [1]. We focus on tables from May 2008 snapshot that consist of information about projects that developers work on, various project categories, and programming languages used in the projects. There are 209,009 developers working on 151,776 projects in May 2008 snapshot. From these 151,776 projects, we extract 354 project categories and 90 different programming languages.

We then do several filtering steps based on some basic intuitions. Our tool makes recommendations based on a developer's past history. If a particular developer has only worked on one or a few projects, the accuracy of our tool could be adversely affected as we have insufficient data to make recommendations. Furthermore, for some projects, there are a lot of developers. Various developers might work on totally different parts of the project and might even be unaware of the participation of many other developers in the same project.

Based on the above intuitions, we only include developers who have more than p projects. Furthermore, we only include projects which have less than v developers. We refer to this process as *database projection*. We denote the projection of database D on developers working on at least p projects containing at most v developers as $Proj_v^p(D)$. In this qualitative study, we set $p = 7$ and $v = 3$. We investigate different values of p and v in Section IV-B2. The resultant projected database, *i.e.*, $Proj_3^7$(D), contains 213 projects. These projects involve 67 developers, 136 project categories, and 27 different programming languages. We then build a DPP graph based on these 213 projects. The resultant DPP graph consists of 376 nodes and 1,076 edges.

We process our DPP graph following the procedure described in Section III and extract top-k recommended developers based on user inputs. We use Matlab to perform backend computation and build the user interface using C#.

A snapshot of the user interface is shown in Figure 4. The

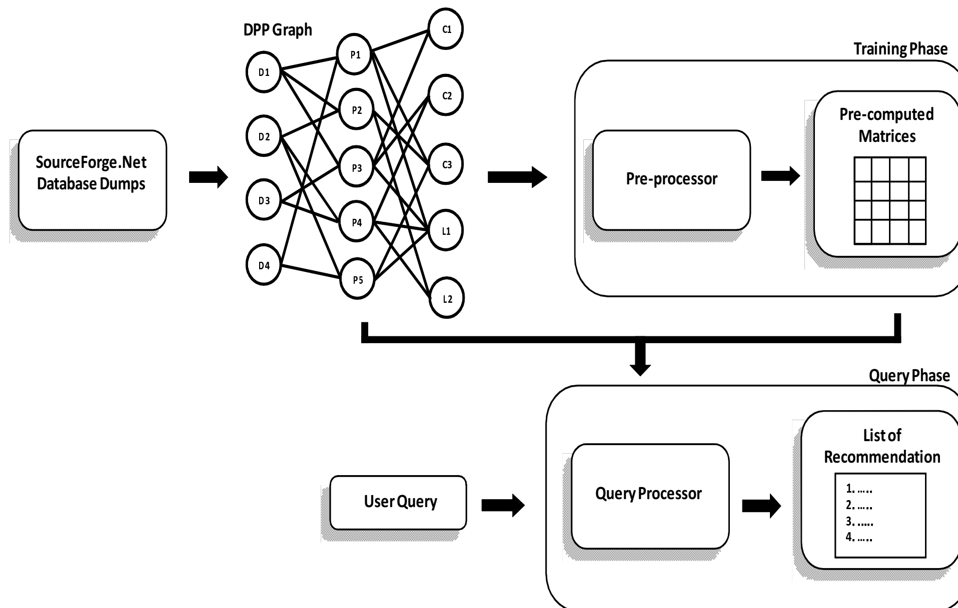

Fig. 3. Overall Recommendation Process

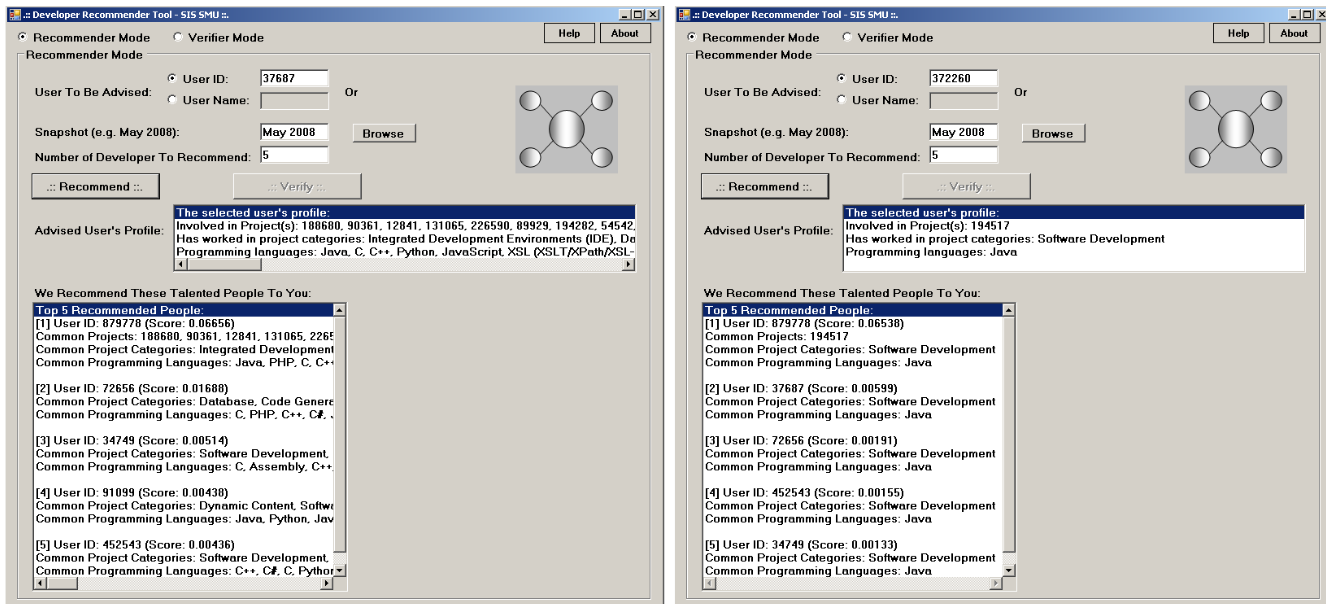

Fig. 4. UI Snapshots: Query 1 (Left) & Query 2 (Right)

tool requests three data inputs, i.e. user ID[2] of a developer requesting recommendations (input developer), information on the database snapshot of SourceForge.Net to be used, and number of recommended developers that will be returned by the tool. After a user presses the "Recommend" button, the tool will output a list of top-k developers based on user's inputs. The tool also outputs what projects, project categories and project programming languages that the recommended developers have in common with the input developer.

Our tool also has a feature to verify whether the top-k recommended developers will create a real collaboration with the query developer or not. This feature enables us to check if our approach is reasonable. We collect multiple SourceForge.Net snapshots. From these multiple snapshots, we check if the query developer really make a collaboration with each top-k recommended developers in the future. The tool outputs the project ID[3] and snapshot where the advised developer really collaborates with the recommended developers. This feature is shown in Figure 6.

2) Runtime & Results: The following describes our findings on querying top-k different developers for recommendations. The runtime needed for the training (only run once) and producing top-k recommendations for various queries are

[2]Every developer in SourceForge.Net has an unique identifier that differentiates one user from the others.

[3]Every project in SourceForge.Net also has an unique identifier.

shown below. We show the runtime for various k values, i.e. 5, 10, 15, and 20.

k Value	Time
5	0.0298s
10	0.0291s
15	0.0297s
20	0.0299s

Next we show our findings on querying 4 different input developers for recommendations. We also present our tool's user interface when showing the results.

Query 1. As the first query, we input a developer with user ID *37687*. Using k equals to 5, our tool recommends developers with user ID *879778, 72656, 34749, 91099,* and *452543*. Developer *879778* is ranked higher than the other developers as it is the only one with some common projects with the input developer. The rest of the recommended developers have common project categories and programming languages with the input developer. Figure 4 shows the result for **Query 1**.

Query 2. As the second query, we input a developer with user ID *372260*. We use k equals to 5 and our tool recommends developers with user ID *879778, 37687, 72656, 452543,* and *34749*. All recommended developers have a common project category with the query developer, i.e. Software Development. They also have a programming skill in Java which is the same with the query developer. Developer *879778* is ranked first as he has worked together with the input developer on a common project, i.e. the project with ID *194517*. Figure 4 shows the result for **Query 2**.

B. Quantitative Results

We first describe our experiment settings and then present the result of our quantitative experiments. We also present a sensitivity analysis to evaluate the effect of varying k (i.e., the number of developers to recommend) on the quality of our recommendations.

1) Experiment Settings: The previous section describes the results of our tool on various queries and shows that the results are reasonable. We would like to also quantitatively evaluate our tool. A recommendation is good if it eventually leads to a collaboration among developers. To evaluate how good our recommendation is we take two consecutive monthly snapshots, say D_o and D_n, of the SourceForge.Net dataset. We then evaluate the proportion of the new collaborations that corresponds to recommendations made by our tool. We use this as a proxy of our recommendation accuracy. In practice, if our tool is deployed, we believe the result would likely to be better as it would *actively* encourage new collaborations.

The following paragraphs describe how our recommender system is trained, how a new collaboration test set is formed, and how the accuracy of the recommender system is evaluated.

Training Recommender System. Similar to the qualitative analysis, we would focus on developers who worked on at least 7 projects and each project has at most 3 developers. We perform database projection on D_o to result in D_{Set} (see Section IV-A1). These are the important projects and developers in D_o. We use this to train our recommender system $REngine$.

Building Collaboration Test Set. Next, we build a test set that represent new collaborations in D_n. A collaboration is a mapping between a developer to a set of other developers he or she works with. A collaboration test set is thus a set of mappings. To extract this new collaboration test set $Collab$, we follow the following steps:

1) Compare D_o and D_n to get a set of new projects NP. A new project is a project that appears in D_n but not in D_o.
2) Find the set of developers $devTest$ in D_{Set} working on one or more projects in NP. For these developers we could use the recommender engine $REngine$ trained on D_{Set}.
3) For each developer d in $devTest$ and for each project p in NP that d worked in, we add the mapping $d \mapsto C$, where C are the collaborators of developer d in project p, to the collaboration test set $Collab$.

Evaluating the Accuracy of Recommender Engine. For each developer $devQ$ in $Collab$ (i.e., each developer in the domains of the mappings in $Collab$), we run $REngine$ which returns a list of recommended developers. We say a recommendation is successful for a new project P that $devQ$ works on if at least one person in the recommended list works with $devQ$ in P. For each collaboration mapping in $Collab$, there are 2 possible cases for our recommendation: our recommendation is successful or our recommendation is not successful. We refer to these two cases as HIT and MISS. Based on these, we simply compute accuracy by the following formula:

$$accuracy = \frac{|HIT|}{|HIT| + |MISS|}$$

Issue with SourceForge.Net dataset. We initially thought that the database dumps of SourceForge.Net collected by Madey [1] grow over time with new projects added to existing ones. However, we find that this is not the case. We analyze database dumps from May 2008 to May 2010. Although the database dumps grow in size, which means the number of projects is increasing from one snapshot to others, we find that not every project from one snapshot still exists in the subsequent snapshots. However we only use projects that still exist from D_o and D_n. The growth of database dumps could also be used as an indicator that many new projects are introduced in the snapshots, which means new collaborations are formed between developers. We show the growth of the SourceForge.Net databases from May 2008 to May 2010 in Figure 5.

The feature *Verifier Mode* in our tool enables users to verify if the list of recommended developers will eventually form a real collaboration with the input developer in a future snapshot. Figure 6 shows our tool in *Verifier Mode*. As the input developer is a developer with user ID *37678*, our tool

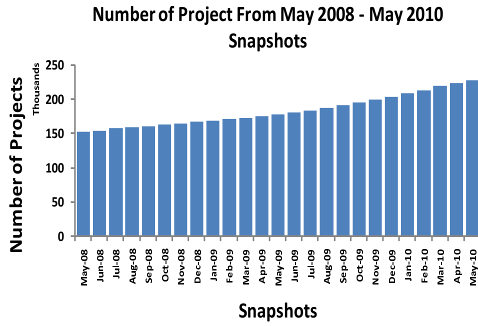

Fig. 5. Number of Projects From May 2008 To May 2010

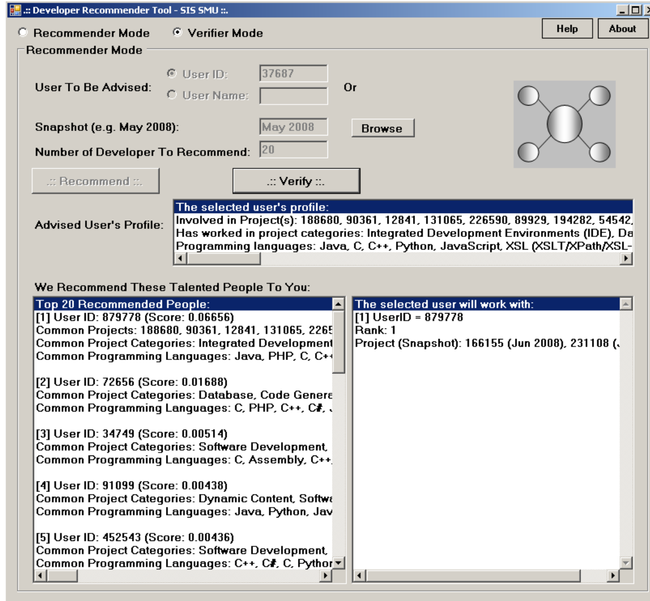

Fig. 6. Result's Verification

recommends developers with user ID *879778, 72656, 34749, 91099,* and *452543*. After clicking "Verify" button, the tool returns a developer with user ID *879778* who is ranked first in the recommendation list. Developer (*879778*) makes future collaborations with the input developer (*37678*) in projects with project ID *166155* and *231108* in June 2008 and July 2008 respectively.

2) Accuracy Results & Sensitivity Analysis: Our approach takes as input a user-defined parameter k which is the number of developers to recommend. Using the value of k equals to 20, for the May 2008 to May 2010 datasets, we could achieve an accuracy of 83.33%. To evaluate the sensitivity of our algorithm on the value of k, we vary this k and evaluate the accuracy of the recommendations. We plot the effects of varying k on accuracy in Figure 7.

The accuracy values ranges from 78.79% to 83.33%. There is only 4.54% reduction in accuracy if we use various k values from 5 to 20. This means that our top few recommendations are successful. The average time for the training phase is 0.03 seconds and the query phase only needs less than a second to make recommendations. This shows that our approach is efficient enough to allow users to perform an interactive query

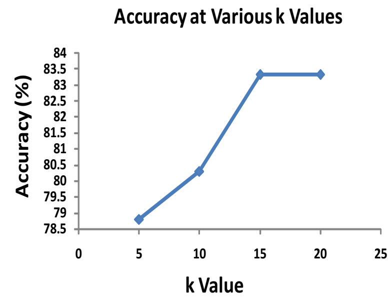

Fig. 7. Accuracy values when varying k

p Value	v Value	Accuracy (%)
5	2	77.16
	3	70.62
6	2	78.57
	3	80.21
7	2	79.17
	3	83.33

TABLE I
ACCURACY VALUES WHEN VARYING p AND v

with the tool.

We also consider different values of p and v for our database projection. The accuracy results for various settings of p and v is shown in Table I. From the results, it can be seen that for various values of p and v our tool's accuracy is reasonable. We must admit when p is very low and v is very high we notice poor accuracy. However, these are cases where our tool has insufficient data to make a reasonable recommendations.

C. Threats to Validity

Similar to other empirical studies, there are several threats to validity in interpreting the results.

Threat to construct validity corresponds to the appropriateness of our evaluation metrics. We evaluate our approach qualitatively by presenting how our tool could be used and quantitatively by comparing the number of future collaborations that match our recommendations. We believe these two evaluation approaches are reasonable. We acknowledge that the evaluation could be strengthened by a large industrial user study involving a developer collaboration network in a large software company. Our system could be deployed for a few months and successful cases of "happy" collaborations could be the evaluation metric. This is a potential future work.

Threat to internal validity corresponds to the ability of our experiments to link the independent variable (i.e., input variable to be varied during the experiment) and the dependent variable (i.e., target variable). Our qualitative evaluation is rather subjective. Experimenter bias is one threat of internal validity. However, our quantitative and sensitivity evaluations are objective. The dependent variable, namely the accuracy measure, is computed based on historical data in SourceForge.Net.

Threat to external validity corresponds to the ability to generalize our results to recommending developers in general. In this study, we experiment with developers involved in

a large open-source software super-repository. The projects range from large to small, popular to unpopular, C to Java, etc. We show that our tool is able to make good recommendations when there is sufficient data. It remains unexplored how one could make reasonable recommendations when there is little data. Also, we have only analyzed open-source software development. Proprietary/closed-source software development might follow a different pattern that might impede the effectiveness of our recommendation system.

V. Discussion

It might seem that our approach could be done by simply performing SQL queries. This is not the case as random walk with restart captures not only direct relationships between developers, project, and project properties such as project categories and programming languages, but also transitive relationships between them. We also have two additional benefits: (1) In the case the SQL queries return empty results, our approach still works by considering the indirect or transitive relationships; (2) In the case, the SQL queries return too many results, our approach can output the top-k since we do a ranking based on random walk with restart algorithm.

We show an example of the case mentioned above in Figure 8. We consider developer *11970* and k set to 20. For these inputs, our tool returns a list of recommended developers including: *405291, 69820, 161, 297846,* and *72656*. After performing verification, our tool shows on the right pane that the input developer makes future collaborations with all these recommended developers[4]. All of these developers, *i.e.*, developer *405291, 69820, 161, 297846,* and *72656*, do not have any common project, project category, and programming language skill with the input developer. However, due to transitive relationships captured by our tool that leverages random walk with restart we are able to recommend successfully.

Sometimes, one might want to look for collaborators/co-developers that posses a particular skill. Our method could be naturally extended to support this kind of query by post-filtering the ranking result by RWR based on a given criteria. Although our current tool does not directly support such constrained query, it could be easily extended to find these developers.

The project properties in our DPP graph are project categories and project programming languages. Although in this work we only use 2 types of project properties to construct DPP graph, it is possible to extend them to include more types. In this work, we do not conduct a real-life survey with feedback from actual developers to decide properties to be used. We assume that in an open-source setting, developers decide to join in or contribute to a particular project motivated by who invites them and the project's details. Regarding the latter one, there is a tendency that project category and project programming language are considered by the developers before they join the project.

[4]Developer with user ID *72656* is not shown on the screen

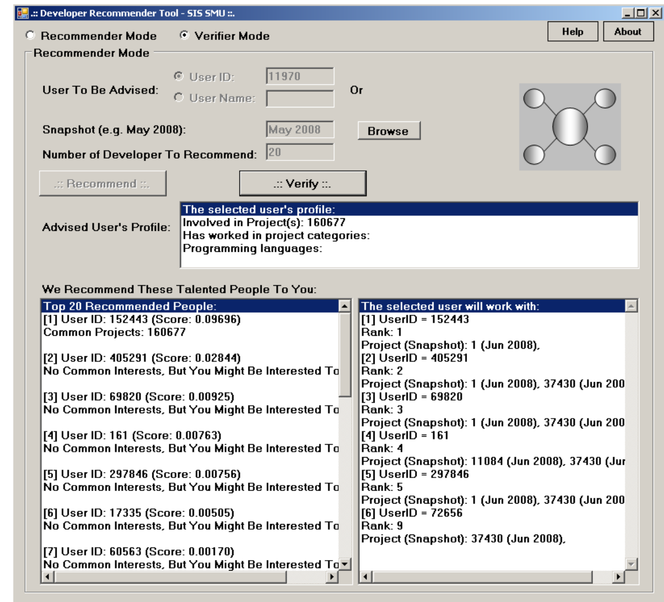

Fig. 8. Transitive Relationships

In this work, we do not differentiate developers' relative contribution in a project. Some developers act as project leaders, others contribute most of the project, while yet others contribute little to the project. We simply take the list of contributors recorded in SourceForge.Net site. Note that we do not look into the various SVN/CVS repositories. We believe one needs to contribute substantially before he or she is listed as one of the contributor in SourceForge.Net site. Furthermore analyzing SVN/CVS for tens of thousands of projects poses a scalability issue. Another issue related to using SVN/CVS information to analyze a super-repository is the uniqueness of the committer names in SVN/CVS. A member of a super-repository could have multiple user names in various SVN/CVS repositories. Furthermore, two members in a super-repository could have the same user name in different SVN/CVS repositories. Indeed, during our preliminary study on a small set of SVN/CVS repositories of frequently downloaded projects in SourceForge.Net, we find that the identifier involved in the most projects is "root" which likely corresponds to more than one developer.

VI. Conclusion and Future Work

In this work, we propose a new approach to recommend people from developers' collaboration network extracted from a super-repository containing numerous developers, projects, and project properties. We represent the collaboration network of developers in the super-repository by a tripartite graph referred to as Developer-Project-Property (DPP) graph. Based on the graph, given an input developer, we compute a measure of compatibility that would rank developers based on the number of common projects, common project properties, and their transitive relationships to the input developer. We find that the similarity measure based on random walk with restart used in search engine to be very appropriate. We build our

solution on top of a fast random walk with restart solution that breaks a large graph into parts and pre-compute inverse matrices of the partial graphs. Each query could then be performed fast based on the pre-computed inverse matrices. To evaluate our proposed recommendation approach, we have performed qualitative and quantitative experiments on a dataset extracted from database dumps of Sourceforge.Net. The experiment shows that the approach could run fast and produce a reasonable list of recommendations. In our quantitative evaluation, we show that our approach could yield up to 83.33% accuracy.

In the future, we plan to extend the evaluation our approach by means of a user study. An industrial user study on a network capturing collaborations among software developers working in various projects of a large corporation would be interesting. No constraint is currently supported in this work; in the future we would like to incorporate constraints into the recommendation system. We plan to develop a query language that would further help users to specify the various constraints on their desired collaborators. Aside from recommending developers, we would also like to investigate the applicability of random walk with restart to produce different types of recommendations based on historical software engineering data.

Acknowledgement. We would like to thank Greg Madey for sharing with us the SourceForge.Net dataset. We would also like to thank Shaowei Wang for valuable discussion and National Research Foundation (NRF) (NRF2008IDM-IDM004-036) for funding the work.

REFERENCES

[1] M. Antwerp and G. Madey, "Advances in the sourceforge research data archive (SRDA)," in *Int. Conf. on Open Source Systems*, 2008.

[2] C. Bird, E. Barr, A. Nash, P. Devanbu, V. Filkov, and Z. Su, "Structure and dynamics of research collaboration in computer science," in *SDM*, 2009.

[3] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *International Working Conference on Mining Software Repositories*, 2006.

[4] M. Cataldo and J. Herbsleb, "Communication networks in geographically distributed software development," in *CSCW*, 2008.

[5] J. Chen, W. Geyer, C. Dugan, M. Muller, and I. Guy, ""make new friends, but keep the old": Recommending people on social networking sites," in *Proceeding of The 27th International Conference on Human Factors in Computing Systems (CHI)*, 2009.

[6] F. Chua and E.-P. Lim, "Trust network inference for online rating data using generative models." in *Proceedings of the 16th International Conference on Knowledge Discovery and Data Mining (KDD)*, 2010.

[7] F. C. de Sousa, N. C. Mendona, S. Uchitel, and J. Kramer, "Detecting implied scenarios from execution traces." in *Proceeding of The 14th Working Conference on Reverse Engineering (WCRE)*, 2007, pp. 50–59.

[8] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history." in *Proceeding of International Conference on Software Maintenance (ICSM)*, 1998.

[9] J. Herbsleb, D. Paulish, and M. Bass, "Global software development at Siemens: Experience from nine projects," in *ICSE*, 2005.

[10] H. Kagdi and D. Poshyvanyk, "Who can help me with this change request?" in *IEEE 17th International Conference on Program Comprehension (ICPC)*, 2009, pp. 273–277.

[11] H. Kagdi, M. Collard, and J. Maletic, "An approach to mining call-usage patterns with syntactic context." in *Proceeding of 22nd International Conference on Automated Software Engineering (ASE)*, 2007, pp. 457–460.

[12] T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *Proceeding of The 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2009.

[13] J. Leskovec and E. Horvitz, "Planetary-scale views on a large instant-messaging network," in *WWW*, 2008.

[14] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," in *Proc. 2005 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'05)*, Chicago, IL, Aug. 2005, pp. 177–187.

[15] M. Lungu, M. Lanza, T. Girba, and R. Heeck, "Reverse engineering super-repositories," in *Proceeeding of The 14th Working Conference on Reverse Engineering (WCRE)*, 2007.

[16] D. Ma, D. Schuler, T. Zimmermann, and J. Sillito, "Expert recommendation with usage expertise," in *IEEE International Conference on Software Maintenance (ICSM)*, 2009.

[17] G. Madey, V. Freeh, and R. Tynan, "The open source software development phenomenon: An analysis based on social network theory," in *AMCIS*, 2002.

[18] T. Nguyen, H. Nguyen, N. Pham, J. Al-Kofahi, and T. Nguyen, "Graph-based mining of multiple object usage patterns." in *ESEC/SIGSOFT FSE*, 2009, pp. 383–392.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd, "Pagerank citation ranking: Bringing order to the web," in *Technical Report, Stanford University*, 1998.

[20] K. Pan, S. Kim, and E. W. Jr., "Toward an understanding of bug fix patterns." *Empirical Software Engineering*, vol. 14, pp. 286–315, 2009.

[21] M. Pinzger, N. Nagappan, and B. Murphy, "Can developer-module networks predict failures ?" in *Proceeding of The 16th International Symposium on Foundations of Software Engineering (FSE)*, 2008.

[22] N. Ramasubbu and R. Balan, "Globally distributed software development project performance: An empirical analysis," in *FSE*, 2007.

[23] S. Rastkar and G. Murphy, "On what basis to recommend: Changesets or interactions?" in *IEEE International Working Conference on Mining Software Repositories (MSR)*, 2009.

[24] F. Ricca and A. Marchetto, "Heroes in floss projects: An explorative study," in *Working Conference on Reverse Engineering (WCRE)*, 2010.

[25] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Proceeding of The 31th International Conference on Software Engineering (ICSE)*, 2009.

[26] M. Shevertalov and S. Mancoridis, "A reverse engineering tool for extracting protocols of networked applications." in *Proceeding of The 14th Working Conference on Reverse Engineering (WCRE)*, 2007, pp. 229–238.

[27] D. Surian, D. Lo, and E.-P. Lim, "Mining collaboration patterns from a large developer network," in *Proceeding of The 17th Working Conference on Reverse Engineering (WCRE)*, 2010.

[28] S. Thummalapenta and T. Xie, "Spotweb: Detecting framework hotspots and coldspots via mining open source code on the web." in *The 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2008, pp. 327–336.

[29] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *Proceeding of The 6th International Conference on Data Mining (ICDM)*, 2006.

[30] A. Wasylkowski and A. Zeller, "Mining temporal specifications from object usage." in *Proceeding of The IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2009, pp. 295–306.

[31] M. Wermelinger, Y. Yu, and M. Strohmaier, "Using formal concept analysis to construct and visualise hierarchies of socio-technical relations," in *International Conference on Software Engineering (ICSE)*, 2009.

[32] T. Wolf, A. Schroter, D. Damian, and T. Nguyen., "Predicting build failures using social network analysis on developer communication." in *Proceeding of The 31st International Conference on Software Engineering (ICSE)*, 2009.

[33] J. Xu, Y. Gao, S. Christley, and G. Madey, "A topological analysis of the open source software development community," in *Proceedings of The 38th Annual Hawaii International Conference on System Sciences (HICSS)*, 2005.

[34] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes." in *Proceedings of The 26th International Conferences on Software Engineering (ICSE)*, 2004.