DevNet: Exploring Developer Collaboration in Heterogeneous Networks of Bug Repositories

Song Wang[1], Wen Zhang[1, 3], Ye Yang[1, 2], Qing Wang[1, 2]
[1]Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences
[2]State Key Laboratory of Computer Science, ISCAS
[3]State Key Laboratory of Software Engineering of Wuhan University
{wangsong,zhangwen,yangye,wq}@nfs.iscas.ac.cn

Abstract—**During open source software development and maintenance, bug fixing is a result of developer collaboration. Understanding the structure of developer collaboration could be helpful for effective and efficient bug fixing. Most prior work on exploring developer collaboration in bug repositories only considers a particular form of developer collaboration. However, in real software bug repositories, developers collaborate with each other via multiple ways, e.g., commenting bugs, tossing bugs, and assigning bugs. In this paper, we present DevNet, a framework for representing and analyzing developer collaboration in bug repositories based on heterogeneous developer networks. Moreover, we illustrate that such developer collaboration can assist bug triage through a case study on the bug repositories of Eclipse and Mozilla involving over 800,000 bug reports. Experiment results show that our approach can improve the state-of-the-art bug triage methods by 5-15% in accuracy. We believe that the proposed approach provides new insights for analyzing software repositories through heterogeneous networks.**

Keywords- developer collaboration; software bug repositories; heterogeneous developer network; bug triage

I. INTRODUCTION

A bug repository that handles and tracks a large number of bugs is widely used during the development and maintenance of many open source software projects. With a bug repository, developers collaborate with each other to improve the quality of software projects by posting problems that they have encountered, making suggestions for resolving bugs, and commenting on existing bugs [1].

In a bug repository, fixing bugs is a team effort, many developers work together to solve problems and during this process they form implicit collaborative developer network [5]. Thus, understanding the structure of developer collaboration in bug repositories could be helpful for building successful software. Along this line, many developer network-related analyses have been proposed in software repositories to deal with problems in real development such as failure prediction [10], [11], developer prioritization [6], bug triage [13], [17] and social structure investigation [5], [7], [8]. Each of these studies applies ideas from social network to construct developer network based on a particular form of developer collaboration (e.g., co-comment bugs between two developers). These developer networks are homogeneous, which has merely one type of node (developers) and one type of link (a particular form of developer collaboration). However, in real usage of software bug repositories, developers collaborate with each other to fix bugs via multiple ways, i.e., developers can discuss how to fix bugs with each other (comment-based collaboration), developers may toss a bug to other developer who can handle this bug well (tossing-based collaboration) [17], and developers may assign new coming bugs to others (assigning-based collaboration). Since a typical social network-based approach cannot deal with multi-relations among nodes [3], in many of existing studies, developer collaboration is simply considered based on one type of the above developer collaboration. Thus, employing social network (in a social network, nodes are entities of the same type, e.g., person; links are relations from the same relation, e.g., friendship) to investigate multiple developer collaboration in bug repositories is not appropriate.

In this paper, we exploit recent advances in heterogeneous network-based analysis of multi-relation to investigate multiple types of developer collaboration among developers in bug repositories. Studies in heterogeneous networks have opened new capabilities in knowledge mining [3], [15], software failures prediction [12], social relations prediction [16], and alzheimer's disease prediction [19].

We present DevNet, a framework to represent and analyze multiple developer collaboration in bug repositories based on heterogeneous developer networks. Different from traditional homogeneous developer network, e.g., Fig. 1 (b), a heterogeneous developer network contains multiple types of entities, such as developers, bugs, comments, components, and products as well as multiple types of links denoting different relations among these entities. Moreover, different paths between two nodes in heterogeneous developer network may denote different relations. For example, in Fig. 1 (a), a path between developers "Tom" and "Jim" as "*Tom-Comment2-Bug Report1- Comment1 -Jim*" means that Tom and Jim have commented on a common bug report. These two developers have also another path as "*Tom-Comment4-BugReport2-Component1-BugReport1-Comment1-Jim*" denotes that they have commented on bug reports belonging to the same component. In order to distinguish the means of different paths between two entities, we propose to use *meta path* (a path that connects entities via a sequence of relations) to denote multi-relation between two entities.

We leverage our approach to improve bug triage in bug repositories. Experiments on Eclipse and Mozilla bug repositories show that our approach can improve the state-of-the-art bug triage methods by 5-15% in accuracy.

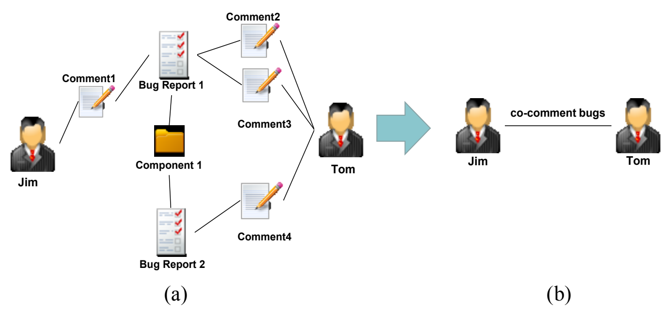


Figure 1. Examples of a simply heterogeneous developer network (a) and a homogeneous developer network (b).

Our contributions can be summarized as follows:

1. We propose a framework to represent, establish, and analyze developer network based on heterogeneous network in bug repositories. To our knowledge, this is the first work for exploring heterogeneous information networks of software repositories.

2. We explore multiple developer collaboration in bug repositories based on heterogeneous developer networks.

3. We examine that developer collaboration revealed in heterogeneous developer networks can improve bug triage in bug repositories.

The rest of this paper is organized as follows. Section II states the background and related work. Section III presents the proposed DevNet framework. Section IV describes the designs of case studies. Section V shows the results of case studies. Section VI states the threats to validity. Section VII concludes this paper and outlines future work.

II. Background and Related Work

A. Bug Triage

Bug triage is a widely known problem during software maintenance, which aims to assign a new bug to a potential developer [1]. Many machine learning based automatic bug assignment algorithms have been proposed to reduce time and labor cost of manual ways. Čubranić et al. [2] model bug triage as a text classification problem. Anvik et al. [1] improve the above work with more extensive preprocessing on the data and more effective classification algorithms. Jeong et al. [17] and Bhattacharya et al. [13] improve bug triage based on tossing graph extracted from tossing histories of bug reports. Other work also refers to this topic, such as developer prioritization based approach [6], and the fuzzy-set and cache-based approach [20].

In this paper, we empirically evaluate whether the developer collaboration revealed in heterogeneous developer networks can improve the result of this problem.

B. Developer Collaboration

Bird et al. [7], [8] examine the collaboration among developers in open source projects by mining communication networks from email archives. Wolf et al. [9] introduce an approach to mine developer collaboration from communication repositories. Additionally, they use their approach to predict software build failures [10]. Bhattacharya et al. [14] employ developer collaboration graph to analyze software evolution. Meneely et al. [4] empirically validate the collaboration relationship among developers can be represented by social network metrics. Based on developer collaboration extracted from comments in bug repositories, [6] proposes a method to prioritize developers and [5] studies the characteristic of developer social networks based on developer collaboration.

Our work differs in two ways from most of these prior studies: (1) we consider developer collaboration in a heterogeneous developer network, which is more complex and with richer information than a homogenous developer network. (2) Taking the advantage of heterogeneous network, we capture more types of developer collaboration.

C. Heterogeneous Network

Heterogeneous Network: A heterogeneous network is defined as a directed graph, in which nodes and relations are of different types [3].

In a heterogeneous network, a meta path is a path consisting of a sequence of relations defined between different entities [15]. Formally, meta path defined as below.

Meta Path: A meta path P is a path defined on a heterogeneous network, and is denoted in form of $O_1 \xrightarrow{R_1} O_2 \xrightarrow{R_2} O_3 \cdots O_{n-1} \xrightarrow{R_{n-1}} O_n$, and O_i ($1 \le i \le n$) is an entity, R_i is a relation between two entities.

Given a specified meta path P: O_1-O_2-$O_3$$\cdots$$O_{n-1}$-$O_n$ in a heterogeneous network, ***random walk*** [3] is frequently used to measure the topological features between two nodes in the meta path, which denotes the probability of the random walk that from x to y (x is an instance of O_1 and y is an instance of O_n) following meta path P. We use $RW_P(x, y)$ to denote the *random walk* from node x to y.

With information explosion in the real world, how to utilize heterogeneous information becomes an important research problem in many areas. For example, [19] improves the prediction of alzheimer's disease by fusing heterogeneous data sources. [12] explores the correlation between heterogeneous developer contribution network and the number of post-release failures. [3] studies the principles and methodologies in mining heterogeneous networks, [15] investigates how to cluster different types of nodes in social network by considering meta path.

In this paper, we represent, establish, and analyze heterogeneous developer networks extracted from bug repositories. Based on heterogeneous developer networks, we investigate multiple types of developer collaboration in bug repositories.

III. DevNet: A Framework to Derive Heterogeneous Developer Network in Bug Repositories

A. Overall Framework of DevNet

Fig. 2 shows the overall structure of DevNet for deriving heterogeneous developer networks in bug repositories. It contains the following four elements:

1) **Bug related Entities**: In a bug repository, there are many types of entities, e.g., developers, documents, software bugs. To fix bugs, these entities frequently interact with each other. Interactions among these entities generate a heterogeneous network.

2) **Schema**: A schema is used to summarize meta structure of a heterogeneous network, which illustrates the entities as well as the relations among entities in a heterogeneous network.

3) **Meta Path**: Meta path is a path, which consists of a sequence of relations between different entities.

4) **Heterogeneous Network**: A heterogeneous network contains multiple types of entities and entities may connect with each other via different meta paths.

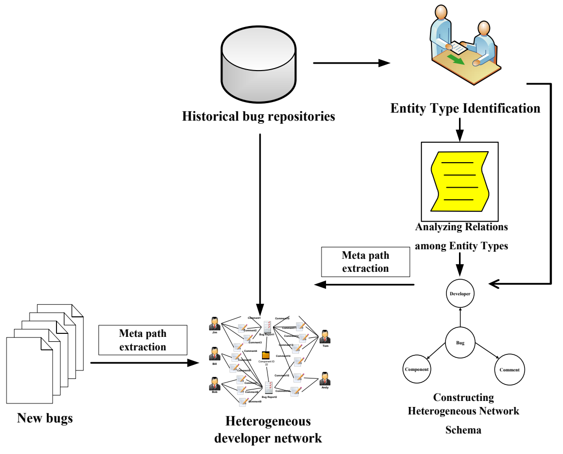

Historical bug repositories
Entity Type Identification
Analyzing Relations among Entity Types
Meta path extraction
Developer
Bug
Component
Comment
Meta path extraction
New bugs
Heterogeneous developer network
Constructing Heterogeneous Network Schema

Figure 2. The overall structure of DevNet.

First, we identify the entities in the bug repositories, and we empirically analyze relations among these entities. Second, we establish a heterogeneous network schema which summarizes the structure of overall heterogeneous network in a bug repository considering the entities and their relations. Third, based on the heterogeneous network schema we use meta path extraction algorithm to extract meta paths between developers to build a heterogeneous developer network. Finally, we update the heterogeneous developer network by adding related meta paths extracted from new coming bugs.

B. Identifying Entities in Bug Repositories

In a typical Bugzilla[1]-based bug repository, such as Eclipse and Mozilla bug repositories, a bug is reported and submitted to the bug tracking system in formation of a bug report, which contains full information of the bug. Thus, developers can comment on the bug report for potential solution of fixing bugs [18]. In a bug repository, each bug belongs to a component of a product. We empirically consider 5 types of entities, namely **developers**[2], **bugs**, **comments, components, and products**. As an abbreviation, we use the first capital letters to denote these entities, namely *D* for developers, *B* for bug reports, *C* for components, and *P* for products. In order to distinguish the abbreviation of components and comments, we use *S* to denote comments.

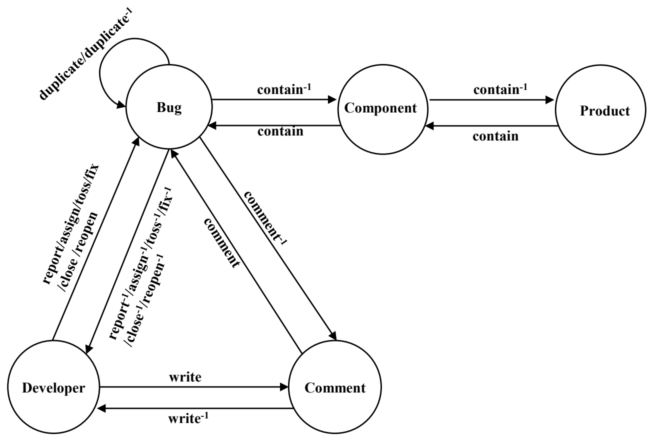


Figure 3. Schema for heterogeneous networks in bug repositories.

C. Analyzing Relations among Entities and Establishing Network Schema

During the software development, entities in bug repositories interact with each other frequently. These interactions denote various relations among entities. We empirically investigate the relations of interactions between entities in a bug repository. During the process of fixing bugs, interactions exist between developers and comments by the relations “write” and “written by” (denoted as *write*$^{-1}$); between bugs and comments by “comment” and “commented by” (denoted as *comment*$^{-1}$); between components and bugs, products and components by “contain” and “belong to” (denoted as *contain*$^{-1}$). Interactions between developers and bugs have multi-relation. Existing work [1], [18] investigate the possible operations from a developer to a bug in a bug repository detailed. Based on their analysis, here we use “*report/assign/toss/fix/close/reopen*” to denote the multiple relations from developers to bugs and “*report*$^{-1}$*/assign*$^{-1}$*/toss*$^{-1}$*/fix*$^{-1}$*/close*$^{-1}$*/reopen*$^{-1}$” to denote the multiple relations from bugs to developers. Interactions exist between bugs by the relations “duplicate” and “duplicate of” (denoted as *duplicate*$^{-1}$).

Having obtained entities and relations among them, we use network schema to summarize the meta structure of a heterogeneous network, which is shown in Fig. 3. In the schema, nodes denote the types of entities, and edges denote

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

[2]Following existing work [5], [6], in this paper, developers include reporters, programmers, testers, and active end users.

relations between entities. Entities can connect with each other via different meta paths, e.g., two developers can be connected via “developer-bug-developer” path, “developer-comment-bug-comment-developer” path and so on.

Note that network schema in Fig. 3 shows overall heterogeneous information of a typical bug repository, i.e., heterogeneous structure between bugs and comments, comments and developers, bugs and developers, and between developers, etc. As a result, to build a heterogeneous developer network in a bug repository, a further work about extracting meta paths among developers is needed.

D. Building Heterogeneous Developer Network

In order to build a heterogeneous developer network in a bug repository, within the network schema, we parse the source data (e.g., bug reports and activity logs) to collect all the meta path instances directly between any two entities, then we build a heterogeneous developer network by extracting meta path instances between developers.

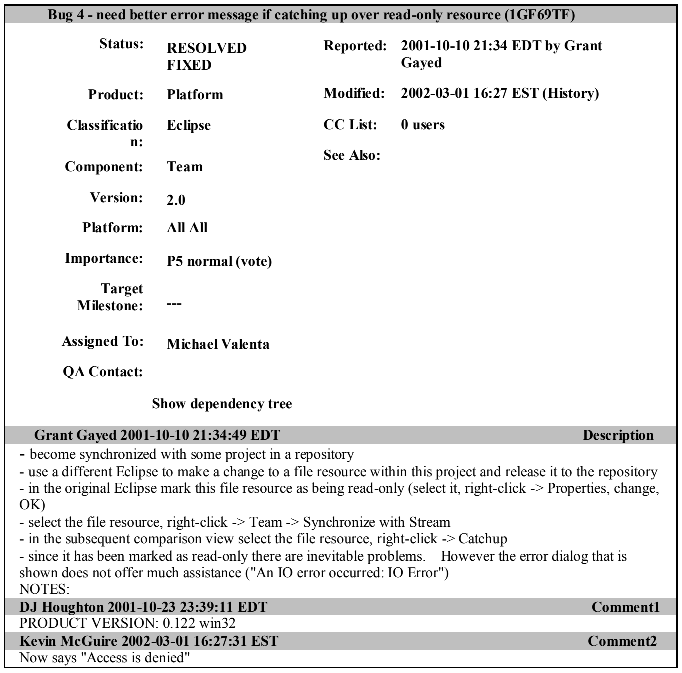
Meta Path Parsing. Typical bug repositories maintain a bug report and an activity log for each bug. A bug report includes information such as the reporter, fixer, commenters and their comments, and the component and product this bug report belongs to. An activity log includes the assigning and tossing histories of a bug report.

In Fig. 4, we take bug report #4 in Eclipse as an example to show the detailed information of a bug report. From the bug report, we can obtain meta path instances between bugs and components (this bug belongs to component “Team”), components and products (component “Team” belongs to “Platform”), developers and bugs (“Grant Gayed” reported this bug and “Michael Valenta” fixed this bug), comments and bugs (comment1 and comment2 are made on this bug), and developers and comments (“DJ Houghton” wrote comment1 and “Kevin McGuire” wrote comment 2).

Table I shows the activity log of bug report #4 in Fig. 4. The bold lines mark the assigning and tossing information of this bug. From the table we can obtain meta path instances between developers and bugs, e.g., “jean-michel_lemieux” assigned this bug to himself and then tossed it to “Kevin McGuire”, later “Kevin McGuire” tossed this bug to “Michael_Valenta”.

Meta paths directly between any two entities in network schema can be obtained by analyzing bug reports and their activity logs. In bug repositories, bug reports and activity logs are typically presented as HTML pages. By crawling and parsing these HTML pages we obtain meta paths directly between two entities.

Building Heterogeneous Developer Network. In a heterogeneous network of a bug repository, there are various meta paths denote different relations between any two entities, to build a heterogeneous developer network, we select meta paths starting and ending with the entity “developers”, e.g., meta path $D \xrightarrow{toss} B \xrightarrow{toss^{-1}} D$ (denotes tossing relation [17]), if there is no ambiguity in either the meaning or the order of the relation this meta path can be abbreviated as *D-B-D*. As a result, 5 types of meta paths in the network schema are selected and presented in Table II, where the meaning of each meta path is given in the second column. After collecting all the directly meta path instances among 5 entities, we use Algorithm 1 to extract instances of meta paths listed in Table II to build a heterogeneous developer network.



Bug 4 - need better error message if catching up over read-only resource (1GF69TF)

Status:	**RESOLVED FIXED**	**Reported:**	**2001-10-10 21:34 EDT by Grant Gayed**
Product:	**Platform**	**Modified:**	**2002-03-01 16:27 EST (History)**
Classification:	**Eclipse**	**CC List:**	**0 users**
Component:	**Team**	**See Also:**	
Version:	**2.0**		
Platform:	**All All**		
Importance:	**P5 normal (vote)**		
Target Milestone:	**---**		
Assigned To:	**Michael Valenta**		
QA Contact:			

Show dependency tree

Grant Gayed 2001-10-10 21:34:49 EDT **Description**

- become synchronized with some project in a repository
- use a different Eclipse to make a change to a file resource within this project and release it to the repository
- in the original Eclipse mark this file resource as being read-only (select it, right-click -> Properties, change, OK)
- select the file resource, right-click -> Team -> Synchronize with Stream
- in the subsequent comparison view select the file resource, right-click -> Catchup
- since it has been marked as read-only there are inevitable problems. However the error dialog that is shown does not offer much assistance ("An IO error occurred: IO Error")

NOTES:

DJ Houghton 2001-10-23 23:39:11 EDT **Comment1**

PRODUCT VERSION: 0.122 win32

Kevin McGuire 2002-03-01 16:27:31 EST **Comment2**

Now says "Access is denied"

Figure 4. Bug report#4[3] in Eclipse Platform.Team.

TABLE I. THE ACTIVITY LOG OF BUG REPORT#4[4]

Who	When	What	Removed	Added
jean-michel_lemieux	**2001-10-12 11:30:02 EDT**	**Assignee**	**Jean-Michel_Lemieux**	**Kevin_McGuire**
		Status	ASSIGNED	NEW
Michael_Valenta	**2001-10-18 16:38:25 EDT**	**Assignee**	**Kevin_McGuire**	**Michael_Valenta**
Michael_Valenta	2001-10-26 12:02:22 EDT	Status	NEW	ASSIGNED
James_Moody	2002-01-03 16:44:14 EST	Priority	P3	P5
Kevin_McGuire	2002-03-01 16:27:31 EST	Status	ASSIGNED	RESOLVED
		Resolution	---	FIXED

TABLE II. SELECTED META PATHS USED FOR BUILDING HETEROGENEOUS DEVELOPER NETWORKS

Meta Path	Meaning of the Relation
$D \xrightarrow{toss} B \xrightarrow{toss^{-1}} D$	developer d_i tosses a bug to developer d_j
$D \xrightarrow{assign} B \xrightarrow{assign^{-1}} D$	developer d_i assigns a bug to developer d_j
D-S-B-S-D	developers d_i and d_j make comments on the common bug report b_i
D-S-B-C-B-S-D	developers d_i and d_j make comments on two bug reports belonged to the same component c_i
D-S-B-C-P-C-B-S-D	developers d_i and d_j make comments on bug reports which belong to two components c_i and c_j in product p_i

[3] https://bugs.eclipse.org/bugs/show_bug.cgi?id=4

[4] https://bugs.eclipse.org/bugs/show_activity.cgi?id=4

Algorithm 1. Meta Path Extraction

```
Input: a heterogeneous network $HN$ in a bug repository, entity instances $O_{start}$ and $O_{end}$, meta path between $O_{start}$ and $O_{end}$ $P$ <$O_{start}$-$O_1$-$O_2$...-$O_k$-...-$O_{end}$> ($1 \le k \le n-1$, n is the length of meta path $P$)

Output:   list $L$ ( instances of meta path $P$ )
1. PathExtract($W$, $O_{start}$, $O_{end}$, $P$,0);

2. Procedure PathExtract($HN$, $O_{start}$, $O_{end}$, $P$,$i$)
3.        $i$=$i$+1; CandidateList= $O_{start;}$
4.        for entity $x$ of $O_i$ do
5.            add $x$ to CandidateList;
6.            if $i$<$n$-1 then
7.                PathExtract($HN$, $O_{start}$, $O_{end}$, $P$,$i$);
8.                else
9.                    add $O_{end}$ to CandidateList;
10.                  add CandidateList to L;
11.             end else;
12.         end if;
13.     end for;
```

TABLE III. SUMMARY OF DATA SETS

Porjects	#Developer	#Component	#Bug report	#Comment	#Product
Eclipse	34562	1200	316911	1670180	140
Mozilla	153577	741	499848	4189149	46

IV. DESIGNS OF THREE CASE STUDIES TO APPLY DEVNET

In this section, we introduce the **designs** of three case studies to apply our proposed DevNet to: 1) build heterogeneous developer networks in bug repositories of both Eclipse and Mozilla; 2) analyze characteristics of heterogeneous developer networks and developer collaboration in these two communities; and 3) illustrate the benefits of DevNet in improving bug triage.

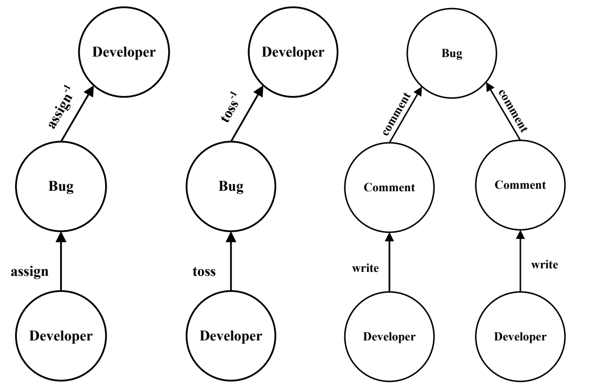
A. *Case Study I: Heterogeneous Developer Networks in Bug Repositories*

In order to examine the proposed DevNet framework, we conduct experiments on two large open source projects: Eclipse and Mozilla. Both projects use Bugzilla as their bug tracking system and both have evolved over 10 years.

Data Set for Case Study I. For Eclipse we collect bug reports from 2001/10/10 to 2010/06/25 (bugs 1-318069) which including 316911 bug reports; for Mozilla we collect bug reports from 1998/04/07 to 2009/09/02 (bugs 35-514157) including 499848 bug reports. The details of our data sets are presented in Table III. Note that since some bug reports are removed during development (e.g., bugs from 346-815 in Eclipse and bugs from 1-34 in Mozilla) or not anonymously accessible (e.g., bug 400020 in Mozilla), the number of collected bugs does not equal to the range of bug IDs.

Building Heterogeneous Developer Networks: For each bug in each project, we collect the meta path instances directly between two entities by parsing its bug report and activity log stored in Eclipse and Mozilla bug repositories. To automate this process, we developed a tool that can automatically crawl the web pages of bug reports and their activity logs, parse meta path instances from these web pages. Then based on the heterogeneous network schema, we use the proposed meta path extraction algorithm (Algorithm 1) to extract meta path instances between developers to build heterogeneous developer networks in both Eclipse and Mozilla.

B. *Case Study II: Analyzing Heterogeneous Developer Networks and Developer Collaboration in Eclipse and Mozilla*



(a) P1: $D \xrightarrow{assign} B \xrightarrow{assign^{-1}} D$ (b) P2: $D \xrightarrow{toss} B \xrightarrow{toss^{-1}} D$ (c) P3: D-S-B-S-D

Figure 5. Three types of meta paths used to represent developer collaboration.

Based on the proposed DevNet framework, we build heterogeneous developer networks in bug repositories of both Eclipse and Mozilla datasets. For better understanding the heterogeneous developer networks, we analyze and compare characteristics of these two heterogeneous developer networks in this case study. Further, based on the heterogeneous developer networks extracted from Eclipse and Mozilla bug repositories, we explore how to represent multiple developer collaboration and study the difference about developer collaboration in Eclipse and Mozilla.

Developer Collaboration via Meta Path: Most of existing work considers developer collaboration based on homogeneous developer networks. However, in real practice of software bug repositories, developers collaborate with each other during the process of fixing bugs via multiple ways. We empirical study three types of developer collaboration and list them as follows:

- ***Assigning-based Developer Collaboration***. For a new coming bug report, a developer of the corresponding component reads the description and assigns the bug to a proper developer to fix it. We address this kind of collaboration by meta path $D \xrightarrow{assign} B \xrightarrow{assign^{-1}} D$, which denotes a developer assigns a bug to other developer.

- ***Tossing-based Developer Collaboration***. When developer d_i is unable to fix a bug assigned to him/her, d_i can toss this bug to other developer who can resolve this bug well. We investigate this kind of developer collaboration via meta path $D \xrightarrow{toss} B \xrightarrow{toss^{-1}} D$.

- ***Comment-based Developer Collaboration***. Fixing bugs is a team work. Many developers may give their suggestions or supply related information by making comments on an existing bug report to help the fixer better resolve the bug. We use meta path D-S-B-S-D to address this kind of developer collaboration in bug repositories.

Fig. 5 presents the three different types of meta paths used in our approach to address multiple developer collaboration in heterogeneous developer networks.

We propose a metric to measure the collaboration probability between two developers to fix bugs in a common component based on different types of developer collaboration in a bug repository.

Given developers d_i and d_j (both i and j range from 1 to n, n is the number of developers) in a bug repository, we denote $CoDev(d_i, d_j, c)$ as the collaboration probability which captures the probability that developer d_i collaborates with d_j to solve bugs in component c. $CoDev(d_i, d_j, c)$ is defined by the following equation based on the *random walk* of three types of meta paths on a given component.

$$CoDev(d_i, d_j, c) = RW_{P1}(d_i, d_j) + RW_{P2}(d_i, d_j) + RW_{P3}(d_i, d_j) \quad (1)$$

Where $RW_{P1}(d_i, d_j)$, $RW_{P2}(d_i, d_j)$, and $RW_{P3}(d_i, d_j)$ denote the *random walk* of three types of meta paths in component c, respectively. Since *random walk* of a meta path ranges from 0 to 1, the range of $CoDev(d_i, d_j, c)$ is [0 3].

C. Case Study III: Improving Bug Triage with Developer Collaboration

Bug triage is a widely known problem during software maintenance, which aims to assign a new coming bug to a potential developer [1]. In this paper, we consider improving bug triage with developer collaboration in heterogeneous developer networks.

Data Set for Bug Triage. We validate our approach on Eclipse and Mozilla projects. Both projects are long-lived and kept stable for more than 10 years. For Eclipse, our data set ranges from bug 200001 to 300000(Aug 2007 to Jan 2010). For Mozilla, we consider bug reports from 400001 to 500000(Oct 2007 to Jun 2009). We use the same heuristics as prior work [1], [6] to remove the non-fixed bug reports (the resolution of bug reports not marked as "fixed") and inactive developers (developers who have fixed less than 50 bug reports). After this, 49539 bug reports of Eclipse and 32097 bug reports of Mozilla are left. We employ *tf-idf* [21], stop words, and stemming to extract string vectors from the title and description of a bug report. For each bug report, the developer who has fixed it is extracted as a label for a machine learning classifier.

Predicting Developers. Bug triage is widely modeled as text classification based on text features extracted from bug reports, following the existing work [6], [13], we employ the incremental learning to evaluate the result of bug triage, we sort bug reports in chronological order and divide them into 11folds and execute 10 rounds to investigate accuracies of all the folds. In each round, we find the *partner* of every developer (*partner* has the biggest possibility to collaborate with this developer) based on developer collaboration in the heterogeneous developer network extracted from the training set, then combine this with the prediction set of a classifier (when using a classifier, the input consists of the string vectors extracted from the title and description of bug reports, and the classifier returns a list of potential developers ranked by relevance) to generate a new prediction set. We use two typical classifiers, i.e., Supporting Vector Machine (SVM) and Naïve Bayes (NB). Formally, we define our prediction approach as follows.

1. Given a new bug report, we predict a list of potential developers by a machine learning classifier, e.g., SVM, NB.

2. Based on the predicted developer set $P = \{p_1, p_2, \ldots, p_n\}$ of an classifier, our approach create a new prediction set, $HP = \{p_1, t_1, p_2, t_2, \ldots, p_n, t_n\}$, t_i is the developer who has the highest probability to collaborate with p_i, also called the *partner* of p_i in this paper.

3. Select the first 5 developers in *HP* to measure our prediction accuracy, the predicted list would be a set, $\{p_1, t_1, p_2, t_2, p_3\}$.

Evaluation. We evaluate the results of our approach with the accuracy of top 2, top 3, top 4, and top 5 predicted developers in the prediction set. The accuracy is defined as $Accuracy = \frac{\#correctly\ predicted\ bug\ reports}{\#all\ the\ bug\ reports}$ based on the recommended prediction set.

V. RESULTS

In this section, we present the results of three case studies designed in Section IV.

A. Results of Case Study I

We apply the proposed DevNet framework to collect meta paths from bug reports and their activity logs. We collect meta path instances between bugs and components (with "*contain/contain*$^{-1}$" relation), components and products (with "*contain /contain*$^{-1}$" relation), developers and bugs (with "*report/report*$^{-1}$" and "*fix/fix*$^{-1}$" relation), comments and bugs (with "*comment/comment*$^{-1}$" relation), developers and comments (with "*write/write*$^{-1}$" relation), and between bugs (with "*duplicate/duplicate*$^{-1}$" relation) from bug reports in each project.

We collect meta path instances between developers and bugs (with "*assign/assign*$^{-1}$", "*toss/toss*$^{-1}$", "*close/close*$^{-1}$", and "*reopen/reopen*$^{-1}$" relations) from the activity logs of bug reports in each project.

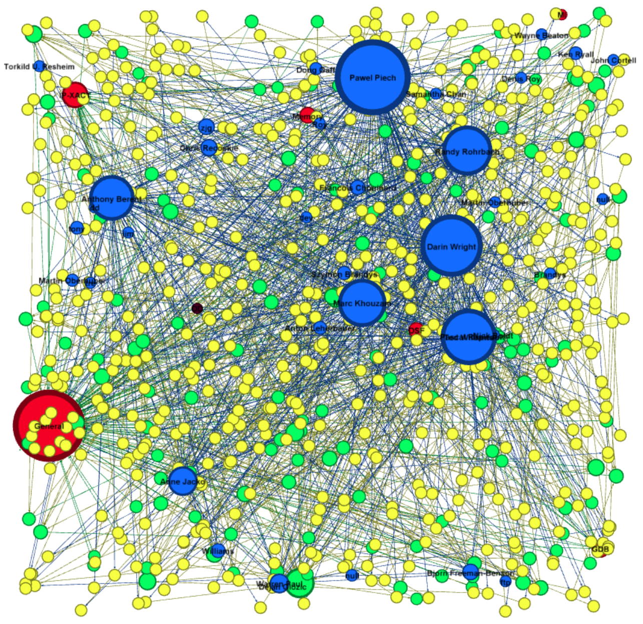


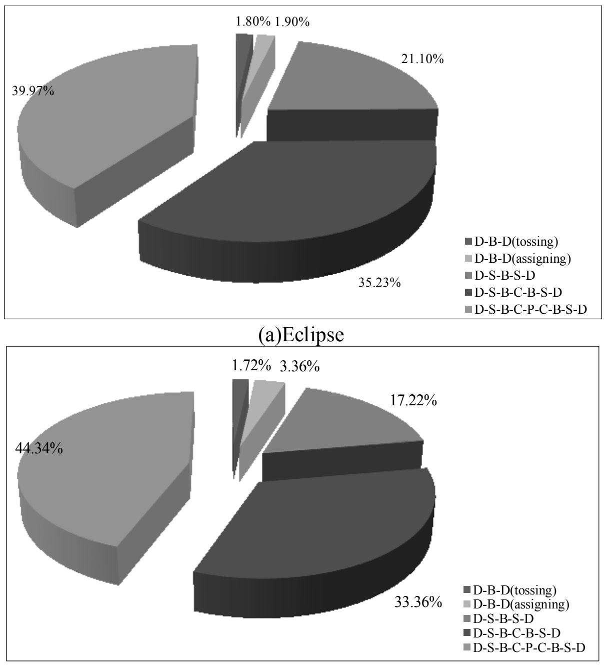
Figure 6. Heterogeneous developer network of product DD (Device Debugging Project) in Eclipse. Nodes in blue denote the developers, in yellow denote the comments, in green denote the bugs, in red denote the components, and in black denote the products.The diameter of a circle denotes the sum of 5 types of meta paths listed in Table II starting from this node. We label developers and componets with their names.

Having obtained all the directly meta path instances (e.g., “developer”-“comment”) among 5 entities, we use Algorithm 1 to extract all the instances of meta paths between developers listed in Table II to build a heterogeneous developer network.

In Fig. 6, we take the product DD (**Device Debugging Project**) in Eclipse as an example to illustrate the heterogeneous network deriving from the proposed DevNet framework. This network contains 38 developers, 152 bugs, 6 components, and 572 comments.

B. Results of Case Study II

In Fig. 7, we present the ratio of 5 types of meta paths (listed in Table II) extracted from bug repositories of Eclipse and Mozilla for building heterogeneous developer networks. In both projects the ratios of meta paths $D \xrightarrow{toss} B \xrightarrow{toss^{-1}} D$ and $D \xrightarrow{assign} B \xrightarrow{assign^{-1}} D$ are lower than other three types of meta paths, and the ratio of meta path D-S-B-C-B-S-D is significant. This implies that most developers interact with each other by commenting bugs of the same component. This fact has been confirmed in prior work [5] that implicit communities may be emerged in bug repositories during software development, and developers of the common communities usually focus on a component. Note that in Eclipse the ratio of meta path $D \xrightarrow{toss} B \xrightarrow{toss^{-1}} D$ and $D \xrightarrow{assign} B \xrightarrow{assign^{-1}} D$ is around 1, however in Mozilla this ratio is up to 2. This means that fixing bugs in Mozilla may take more labor cost than that in Eclipse. One possible reason for this is that products in Mozilla are more complex than those in Eclipse.



(a)Eclipse

(b)Mozilla

Figure 7. The ratio of 5 types of meta paths in Eclipse and Mozilla.

Based on the heterogeneous developer network extracted from Eclipse dataset, we take five developers in component PDE_UI (**Plugin Development Environment UI**) of Eclipse as an example to illustrate how to represent collaboration between two developers. Statistical information about the five developers in component PDE_UI is shown below in Table IV.

TABLE IV. INFORMATION OF FIVE DEVELOPERS IN PDE_UI

Developer	# toss bugs	# assign bugs	# comment bugs
ankur_sharma	17	40	403
cwindatt	50	198	2898
bcabe	1	34	712
caniszczyk	6	345	5407
olivier_thomann	4	119	297

TABLE V. SUMMARY OF THREE TYPES OF META PATH AMONG FIVE DEVELOPERS

Developer	ankur_sharma			cwindatt			bcabe			caniszczyk			olivier_thomann		
	RW_{P1}	RW_{P2}	RW_{P3}	RW_{P1}	RW_{P2}	RW_{P3}	RW_{P1}	RW_{P2}	RW_{P3}	RW_{P1}	RW_{P2}	RW_{P3}	RW_{P1}	RW_{P2}	RW_{P3}
ankur_sharma		-		**0.740**	**0.076**	**1.000**	0.007	0.312	0.000	0.191	0.575	0.000	0.062	0.037	0.000
cwindatt	0.086	0.102	0.889		-		0.214	0.058	0.111	**0.649**	**0.773**	**0.000**	0.051	0.067	0.000
bcabe	0.166	0.306	0.000	0.004	0.043	0.000		-		**0.797**	**0.577**	**0.000**	0.033	0.074	0.000
caniszczyk	**0.518**	**0.646**	**0.000**	0.044	0.090	1.000	0.327	0.161	0.000		-		0.111	0.103	0.000
olivier_thomann	0.226	0.303	0.000	0.079	0.031	0.000	0.076	0.111	0.000	**0.619**	**0.555**	**0.000**		-	

In Table V, we calculate the *random walk* of three types of meta paths among five developers in component PDE_UI. RW_{P1}, RW_{P2}, and RW_{P3} represent the *random walk* of meta paths *P1*, *P2*, and *P3* (shown in Fig. 5) respectively. Column in bold of each row denotes developer in this row has a higher collaboration probability with developer in the column than other four developers, e.g., **ankur_sharma** has higher probability to collaborate with **cwindatt** than others and we call **cwindatt** is the *partner* of **ankur_sharma** in component PDE_UI of Eclipse.

For better understanding the developer collaboration revealed in heterogeneous developer networks of Eclipse and Mozilla bug repositories, we apply our approach to all the developers in Eclipse and Mozilla during 2002/01/01-2008/12/31. Following [6] we choose half year as time interval. We denote the first half year with “f” and the second half year with “s”. For each time interval, we investigate the ratio of developers who change their ***partner*** in the whole projects.

As shown in Fig. 8, the average ratio in Eclipse is 75.7% and in Mozilla is 80.8%. This reflects that developers and teams in Eclipse are more stable than those in Mozilla. Overall, the ratios in both projects are more than 60%. One possible reason for this fact is the unstable developers in the open source projects. Moreover, the ratios after 2004s in Eclipse and after 2005s in Mozilla are fluctuant decreasing. This may because developers and teams are becoming more stable along with the development of these two projects.

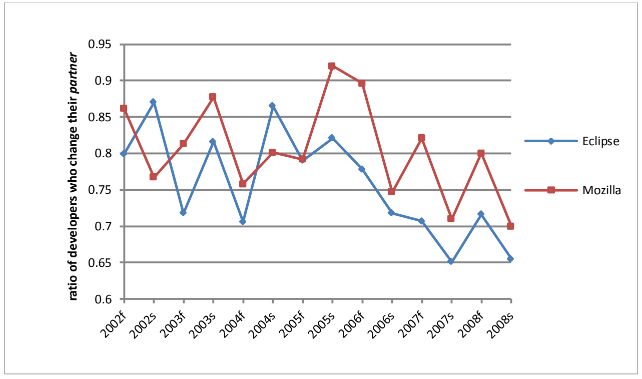


Figure 8. Developer collaboration evolution.

C. Results of Case Study III

In Table VI, we present the results of bug triage by combining developer collaboration with the output of the classifiers (denoted as SVM+MDC or NB+MDC, MDC stands for **M**ultiple **D**eveloper **C**ollaboration) in Eclipse and Mozilla. Overall, for both SVM and Naïve Bayes, the accuracy is improved when combining with developer collaboration. The maximum prediction accuracy for Eclipse is 71.69% and for Mozilla is 58.60%. Note that, the average improvement for Eclipse is around 21%, while for Mozilla, it’s only about 10%. One of the possible reasons for this fact is that the developers and teams in Eclipse are more stable than that in Mozilla. Thus, for Mozilla, developer collaboration extracted from a training data set is very different from that in a testing data set. While in Eclipse, the change of developer collaboration is not as drastic as that in Mozilla so the performance of our approach is better in Eclipse.

TABLE VI. PREDICATION ACCURACY OF BUG TRIAGE ON ECLIPSE AND MOZILLA

Project	classifier	Size	Approach	Accuracy for each fold (%)										Average Accuracy	Improvement
				2	3	4	5	6	7	8	9	10	11		
Eclipse	SVM	Top2	**SVM**	21.85	24.00	27.80	31.69	33.02	28.73	34.26	36.98	37.34	37.47	31.31	23.55
			SVM+MDC	46.77	51.51	49.56	53.08	57.90	58.37	55.86	57.13	59.79	58.62	54.86	
		Top3	**SVM**	25.52	29.68	33.90	37.97	38.63	35.10	41.98	43.38	44.29	44.60	37.51	22.33
			SVM+MDC	49.50	56.37	54.40	58.47	63.59	64.19	61.88	62.46	64.21	63.31	59.84	
		Top4	**SVM**	28.54	33.86	37.48	42.08	43.43	40.28	47.60	48.40	48.78	49.89	42.03	21.64
			SVM+ MDC	52.32	59.72	57.95	61.60	67.68	68.54	66.10	66.47	68.89	67.44	63.67	
		Top5	**SVM**	30.93	37.26	41.16	45.88	47.47	43.72	52.22	52.42	52.93	54.44	45.84	20.72
			SVM+MDC	54.59	62.28	60.75	64.40	70.71	71.27	69.56	69.60	**71.69**	70.60	66.56	
	Naïve Bayes	Top2	**NB**	32.93	34.55	35.44	38.78	41.47	35.61	38.17	38.45	40.85	41.87	37.81	21.65
			NB+MDC	52.81	57.62	55.91	58.72	63.37	63.28	59.44	58.73	62.61	62.07	59.46	
		Top3	**NB**	34.98	36.32	37.19	40.71	43.03	37.21	39.94	41.27	43.29	44.40	39.83	23.47
			NB+MDC	55.98	61.55	58.95	62.36	67.74	66.74	63.77	62.92	66.61	66.33	63.30	
		Top4	**NB**	35.84	36.79	37.90	41.57	43.91	37.94	40.47	42.38	44.32	45.51	40.66	24.31
			NB+MDC	57.92	62.94	60.50	63.94	69.24	68.05	65.39	65.12	68.21	68.36	64.97	
		Top5	**NB**	36.53	37.01	38.32	41.95	44.23	38.30	40.72	42.98	44.85	46.22	41.11	24.67
			NB+MDC	58.87	63.37	61.26	64.67	69.85	68.87	66.25	66.30	69.07	69.33	65.78	
Mozilla	SVM	Top2	**SVM**	18.88	24.09	23.17	22.48	26.42	28.55	29.58	30.84	34.30	32.46	27.08	15.28
			SVM+MDC	40.85	41.16	40.99	40.54	41.23	43.63	43.15	42.97	44.52	44.57	42.36	
		Top3	**SVM**	22.79	29.71	28.72	28.34	33.65	35.61	38.14	38.07	41.43	39.01	33.55	13.71
			SVM+MDC	43.28	45.34	45.41	45.54	46.13	49.04	50.07	48.32	49.42	50.09	47.26	
		Top4	**SVM**	26.38	33.93	33.52	33.24	39.44	41.74	43.59	42.73	46.84	44.60	38.60	13.02
			SVM+MDC	45.85	49.07	49.93	49.31	50.99	53.39	55.00	53.46	54.35	54.82	51.62	
		Top5	**SVM**	29.78	37.77	37.59	37.83	43.35	46.68	48.63	46.57	50.71	48.10	42.70	12.27
			SVM+MDC	47.74	52.23	53.05	52.47	54.49	57.44	**58.60**	57.33	58.15	58.18	54.97	
	Naïve Bayes	Top2	**NB**	33.04	35.16	33.93	32.04	35.06	35.54	38.62	38.49	38.07	36.10	35.61	10.72
			NB+MDC	45.78	47.53	46.20	45.51	45.61	47.29	47.33	46.57	46.40	45.05	46.33	
		Top3	**NB**	35.98	38.59	36.77	35.23	38.21	38.90	41.74	41.30	40.88	39.49	38.71	11.59
			NB+MDC	49.55	51.30	49.73	49.49	49.76	51.68	51.47	50.65	50.55	48.82	50.30	
		Top4	**NB**	37.94	40.34	38.45	36.77	39.86	40.92	43.04	42.67	42.43	40.42	40.28	12.05
			NB+MDC	51.20	53.46	51.88	51.47	51.95	53.29	53.50	52.91	52.78	50.84	52.33	
		Top5	**NB**	38.86	41.19	38.93	37.25	40.61	41.60	43.76	43.15	43.04	41.06	40.95	12.44
			NB+MDC	52.26	54.52	53.05	52.60	53.08	54.32	54.87	54.28	53.63	51.32	53.39	

Comparisons. To further evaluate the effectiveness of our approach, we compare it with the following two similar methods:

- ***Comment-based Developer Prioritization***. In [6], the authors prioritize developers based on their interaction via comments in bug repositories. They examine that their model can improve bug triage by combining the developer prioritization with the output of machine learning classifiers (SVM and NB). We use DP to denote this method.

- ***Tossing Graph.*** In [17], Jeong et al. first introduce the concept of tossing graph in a bug repository. They proposed a method to improve bug triage by combining tossing graph with the output of machine learning classifiers (SVM and NB). Then Bhattacharya et al. [13] improve this method by fine-grained incremental learning and multi-feature tossing graphs. We use TG to denote method in [13].

For the above two methods, we use the same data set as our approach to evaluate their performance. Results are shown in Fig. 9. In the figure, the vertical axis is the accuracy rate, the horizontal axis is the top N list's size (starting from N=2). Results show that, comparing with the other two methods our approach has 5-15% relative improvement in Eclipse dataset, 1-5% relative improvement in Mozilla dataset. Moreover, our approach is more stable than the other two methods in both Eclipse and Mozilla. For example, in Eclipse dataset, based on both SVM and NB the average improvement of our approach is over 20%. However, for method DP and TG, the performance based on SVM is more significant than that based on NB.

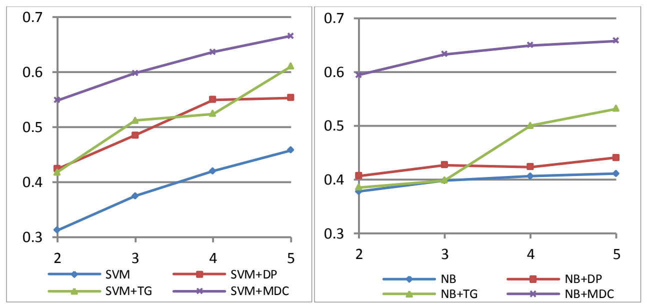
Compared to other two methods, the improvement achieved in our approach is mainly due to the abundant and actual information about developer collaboration revealed by analyzing heterogeneous developer network in bug repositories.

VI. THREATS TO VALIDITY

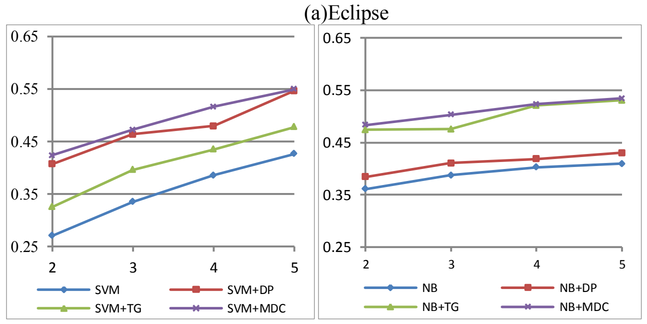
In this section, we discuss the main threats to validity of our approach.

A. *Building Heterogeneous Developer Network*

In this work, we build heterogeneous developer networks based on meta paths between developers extracted from bug reports and bugs activity logs in bug repositories of two large open source projects namely, Eclipse and Mozilla. Both projects maintain a bug report and an activity log for each bug, and most of these documents are anonymously accessible. However, for closed-source software projects (e.g., commercial software projects), entities in their bug repositories and interactions among entities may be different with that of open source projects. Whether our model is suitable for closed-source software should be further investigated.



(a)Eclipse



(b)Mozilla

Figure 9. The Comparisons between our approach with other methods. The vertical axis is the accuracy rate, and the horizontal axis is the top N list's size (starting from N=2).

B. *Developer Collaboration Analysis*

In our work, we consider the developer collaboration based on three types of meta path, however, there are other types of collaboration which are not recorded in software bug repositories and cannot be captured by meta path-based approach, for example a developer may collaborate with others to fix bugs via emails or offline table meetings. Thus, it is hard to validate our obtained results. A further study about examining meta path-based developer collaboration on other kinds of software repositories (mailing list repositories, change log repositories, and source file repositories) is needed.

C. *Improving bug triage*

In this paper we show that the multiple developer collaboration in a heterogeneous developer network is effective to improve bug triage in software repositories based on empirical evaluation. The multiple types of developer collaboration can supply more information to update the output of classifiers. However, further questions may be proposed, for example, why heterogeneous developer network is helpful for bug triage? And what is the internal relation between developer collaboration and fixing bugs? In this paper, we reveal different types of collaboration between developers during the process of fixing bugs. For further work, more case studies should be conducted to explore the correlation between the developer collaboration and bug triage.

VII. CONCLUSION AND FUTURE WORK

During software development and maintenance, fixing bugs is the result of developer collaboration. Thus, understanding the structure of developer collaboration could be helpful for effective and efficient bug fixing. In this paper, we present DevNet, a framework for representing and analyzing developer collaboration in bug repositories based on heterogeneous developer networks. Moreover, we apply such developer collaboration to improve bug triage. Experiments on bug repositories of Eclipse and Mozilla show that our method can improve the state-of-the-art bug triage methods by 5-15% in accuracy. We believe that the proposed approach provides new insights for analyzing software repositories through heterogeneous networks. Our future work consists of the following.

- Exploring the proposed DevNet framework on a variety of projects, including commercial projects. Currently, we only examine our framework on two open source projects. In the future we plan to investigate the performance of DevNet on some closed-source software projects.

- In this paper, we study developer collaboration and its application in heterogeneous developer networks extracted from bug repositories. We are planning to explore other kinds of behaviors of developers in a heterogeneous developer network, e.g., developer contribution, developer communication.

- In this paper, we empirically investigate heterogeneous developer network in bug repositories of software products. A further study about examining heterogeneous developer networks in other kinds of software repositories, for example, mailing list repositories, change log repositories, and source file repositories will be conducted.

To our knowledge, this study is the first to investigate multiple types of developer collaboration by meta path-based approach in heterogeneous developer networks extracted from software bug repositories.

ACKNOWLEDGMENT

This research was supported in part by National Natural Science Foundation of China under Grant Nos. 61073044, 71101138 and 61003028; National Science and Technology Major Project under Grant Nos. 2012ZX01039-004; Beijing Natural Science Fund under Grant No.4122087; State Key Laboratory of Software Engineering of Wuhan University.

REFERENCES

[1] J. Anvik, L. Hiew, and G.C. Murphy, "Who Should Fix This Bug?," Proc. 28th Intl. Conf. Software Engineering (ICSE '06), May 2006,pp. 361-370.

[2] D. Čubranić and G.C. Murphy, "Automatic Bug Triage Using Text Categorization," Proc. 16th Intl. Conf. Software Engineering & Knowledge Engineering (SEKE '04), Jun. 2004, pp. 92-97.

[3] Y. Sun and J. Han: Mining Heterogeneous Information Networks: Principles and Methodologies. Synthesis Lectures on Data Mining and Knowledge Discovery, Morgan & Claypool Publishers 2012.

[4] A. Meneely and L. Williams, "Socio-Technical Developer Networks: Should We Trust Our Measurements?," Proc. 33rd Intl. Conf. Software Engineering (ICSE '11), May 2011, pp. 281-290.

[5] Q. Hong, S. Kim, S.C. Cheung, and C. Bird, "Understanding a Developer Social Network and its Evolution," Proc. 27th IEEE Intl. Conf. Software Maintenance (ICSM '11), Sept. 2011, pp. 323-332.

[6] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer Prioritization in Bug Repositories," Proc. 34st Intl. Conf. Software Engineering (ICSE '12), June.2012, pp. 25-35

[7] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in MSR'06.

[8] C. Bird, D. Pattison, R. D'Souza, V. Filkov and P. Devanbu, "Latent Social Structure in Open Source Projects," Proc. 16th ACM SIGSOFT Intl. Symp. Foundations of software engineering (FSE '08), Nov. 2008, pp.24-35.

[9] T. Wolf, A. Schröter, D. Damian, L. D. Panjer, and T. H. D. Nguyen. "Mining Task-Based Social Networks to Explore Collaboration in Software Teams," IEEE Software, vol. 26, no. 1,pp. 58-66, 2009.

[10] T. Wolf, A. Schröter, D. Damian, and T. Nguyen, "Predicting Build Failures Using Social Network Analysis on Developer Communication," Proc. 31st Intl. Conf. Software Engineering (ICSE '09), May 2009, pp. 1-11.

[11] A. Meneely, L. Williams, W. Snipes, and J. Osborne, "Predicting Failures with Deverloper Networks and Social Network Analysis," Proc. 16th ACM SIGSOFT Intl. Symp. Foundations of Software Engineering (FSE '08), Nov. 2008, pp. 13-23.

[12] M. Pinzger, N. Nagappan, and B. Murphy, "Can Developer-Module Networks Predict Failures?," Proc. 16th ACM SIGSOFT Intl. Symp. Foundations of Software Engineering (FSE '08), Nov. 2008, pp. 2-12.

[13] P. Bhattacharya and I. Neamtiu, "Fine-Grained Incremental Learning and Multi-Feature Tossing Graphs to Improve Bug Triaging," Proc. 26th IEEE Intl. Conf. Software Maintenance (ICSM '10), Sept. 2010, pp. 1-10.

[14] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-Based Analysis and Prediction for Software Evolution," Proc. 34st Intl. Conf. Software Engineering (ICSE '12), June.2012, pp. 419-429.

[15] Y. Sun, B. Norick, J. Han, X. Yan, P. Yu, and X. Yu, "Integrating meta-path selection with user-guided object clustering in heterogeneous information networks," Proc. 18th ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining (KDD'2012), Aug.2012, pp. 1348-1356.

[16] Y. Yang, N. Chawla, Y. Sun, and J. Hani, "Predicting Links in Multi-relational and Heterogeneous Networks," Proc. 12th Int. Conf. Data Mining (ICDM' 12), Dec.2012, pp. 755-764.

[17] G. Jeong, S. Kim, and T. Zimmermann, "Improving Bug Triage with Tossing Graphs," Proc. 17th ACM SIGSOFT Symp. Foundations of Software Engineering (FSE'09), Aug. 2009, pp. 111-120.

[18] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, and C. Weiss, "What Makes a Good Bug Report?," IEEE Trans. Software Engineering, vol. 36, no.5, Oct. 2010, pp. 618-643.

[19] J. Ye, K. Chen, T. Wu, J. Li, Z. Zhao, R. Patel, M. Bae, R. Janardan, H. Liu, G. Alexander, and E. Reiman E, "Heterogeneous data fusion for alzheimer's disease study," Proc. 14th ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining (KDD'2008), July.2008, pp. 1025–1033.

[20] A. Tamrawi, T.T. Nguyen, J.M. Al-Kofahi, and T.N. Nguyen, "Fuzzy Set and Cache-Based Approach for Bug Triaging," Proc. 19th ACM SIGSOFT Symp. Foundations of Software Engineering (FSE '11), Sept. 2011, pp. 365-375.

[21] I.H. Witten, E. Frank, and M.A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 3rd ed. Morgan Kaufmann, Burlington, MA, 2011.