SCIENCE CHINA
Information Sciences

• REVIEW •

December 2014, Vol. 57 121101:1–121101:23
doi: 10.1007/s11432-014-5221-6

Developer social networks in software engineering: construction, analysis, and applications

ZHANG WeiQiang[1], NIE LiMing[2], JIANG He[2], CHEN ZhenYu[1] & LIU Jia[1]*

[1]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093*, China;*
[2]*School of Software, Dalian University of Technology, Dalian* 116024*, China*

Received August 29, 2014; accepted October 8, 2014; published online October 29, 2014

Abstract With the increasing popularity of Internet, more and more developers are collaborating together for software development. During the collaboration, a lot of information related to software development, including communication and coordination information of developers, can be recorded in software repositories. The information can be employed to construct Developer Social Networks (DSNs) for facilitating tasks in software engineering. In this paper, we survey recent advances of DSNs and examine three fundamental steps of DSNs, namely construction, analysis, and applications. We summarize the state-of-the-art methods in the three steps and investigate the relationships among them. Furthermore, we discuss the main issues and point out the future opportunities in the study of DSNs.

Keywords developer social networks, social network analysis, software maintenance, communication and coordination, prediction and recommendation

Citation Zhang W Q, Nie L M, Jiang H, et al. Developer social networks in software engineering: construction, analysis, and applications. Sci China Inf Sci, 2014, 57: 121101(23), doi: 10.1007/s11432-014-5221-6

1 Introduction

With the popularity of Internet, more and more developers geographically distributed in the world are getting involved in one software project [1–3]. Such projects usually have software requirements changing dynamically, agile software process evolving rapidly, teams organized loosely, and developers collaborating freely. In the context, the activities of developers could be recorded in some software repositories, e.g., email systems, configuration management tools, bug tracking systems, etc. [4], in order to facilitate coordination and communication among developers.

It should be noted that in traditional software development, the number of developers may be considered as a negative factor, i.e. adding developers into a team may slow down the progress further [5]. In contrast, there are many new features in modern software development nowadays. First, as both the size and the complexity of software increase, the number of developers in a software team also grows. Second, developers' collaboration methods are mainly those based on Internet instead of face-to-face talk. Third, developers have more chances to participate in tasks of different software development phases. Therefore, developers and relationships among developers take important roles in modern software development. It becomes a new challenge to utilize the information of developers to improve software engineering tasks.

In the past decade, a series of studies have been conducted by using the method of Developer Social Networks (DSNs)[1)] to facilitate tasks in software engineering. In this paper, we review recent advances

* Corresponding author (email: liujia@software.nju.edu.cn)
1) http://software.nju.edu.cn/iSE/DSN.

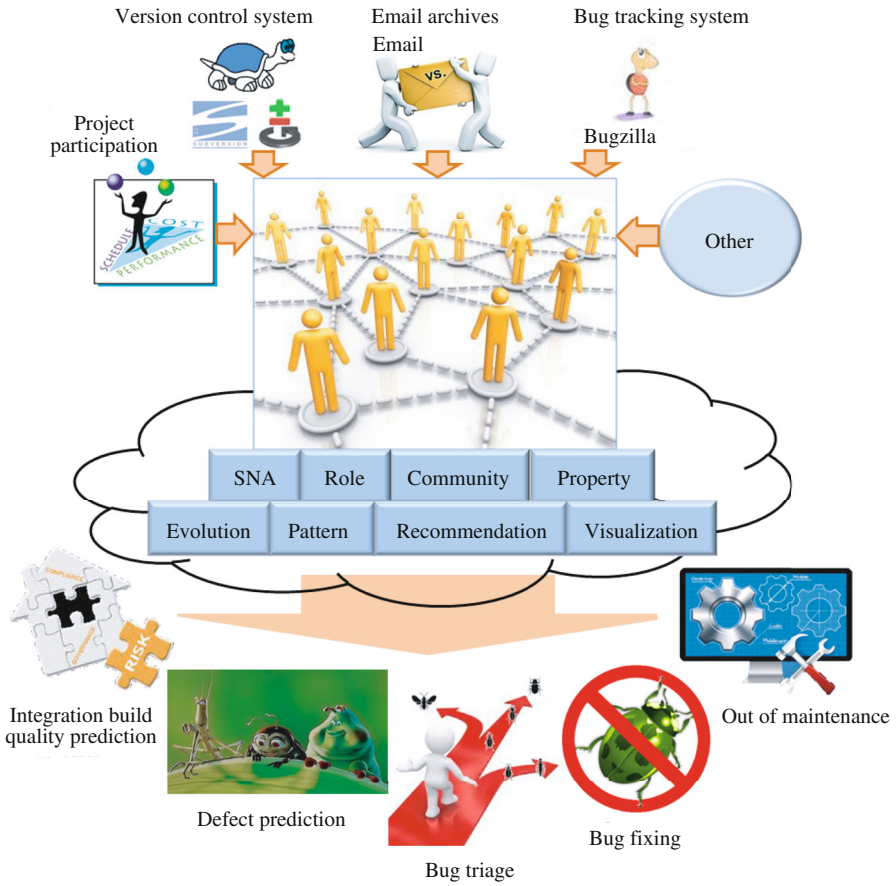

Figure 1 Construction, analysis and applications of DSNs.

related to DSNs and divide the studies of DSNs into three steps: construction, analysis, and applications. The main purpose of this paper is to examine the performance of these steps and the relationships hidden behind them.

The construction methods, analysis methods, and application scenarios of DSNs are briefly shown in Figure 1. DSNs can be constructed by different methods based on different data sources and developer relationships. DSNs can be analyzed by various social network analysis methods that capture and understand the features and characteristics of DSNs. These features and characteristics of DSNs will be finally applied to certain software engineering tasks, in order to make them effective and efficient.

In this paper, we collect all the related papers with DSNs in the literature (the methodology used to get the set of papers is described in Section 2), and summarize DSN construction methods (Section 3), DSN analysis methods (Section 4), and DSN application scenarios (Section 5) respectively. Then we discuss the main issues of DSNs in Section 6, including why DSNs work, when DSNs can be used, and how to use DSNs. Finally, we point out the future research directions about DSNs in Section 7, including new construction methods, new analysis methods, and new application scenarios.

2 Paper selection methodology

First of all, we need to collect papers related with our topic, i.e. developer social networks. The methodology is as follows. The collection process starts from Google Scholar[2)]. We search research papers using the keywords "developer network" and check the first 50 search results. Then unrelated papers are fil-

2) http://scholar.google.com.

tered out. We usually just read the title and abstract. But if necessary, we also take a brief look at the introduction part.

In order to filter out papers, three criteria are used. First, if the paper does not discuss topics about software engineering, it is excluded. For example, many found papers are about general social networks, and they are out of our scope. Second, if there are no networks constructed in the paper, it is excluded. For example, some papers discuss communication in software development, but do not construct any developer networks. Third, if nodes of the network in the paper are not developers, such as source files, the paper is excluded. After a round of selection, there are no more than twenty papers left.

From these papers, we get their references and the papers citing them. Then the above three filtering steps are performed again on the result papers. This iterative process does not stop until there are no more new papers selected. In the end, we collect 86 related papers.

3 Construction of developer social networks

In this section, we will summarize the construction methods of DSNs in our paper collection to investigate the current research status. DSNs constructed by distinct methods may lead to different conclusions, and finally can be applied in different software engineering tasks.

DSNs describe developers and relationships among them in the form of networks. A fundamental problem in constructing DSNs is the definition of relationships among developers. Many kinds of relationships will arise among developers in the life-cycle of software development. Developers need to communicate, collaborate, or coordinate with one another when they are performing different tasks in software development.

Nowadays, the Internet has become available to everyone all over the world. This makes open-source software development more and more popular. Anyone interested in a project can participate in it as a developer[3)], such as a bug reporter, a tester, a commenter, or even a code committer. As a result, there are a large number of developers involved in one project with many communication and coordination links. These links are left in software repositories latently. It is natural to use these links to improve software engineering tasks. As the first step, DSNs can be constructed with these links in the software repositories.

With respect to the data source of software repository, the existing methods for constructing DSNs can be divided into five categories: project participation based DSNs (PP-DSNs), version control system based DSNs (VCS-DSNs), email archives based DSNs (EA-DSNs), bug tracking system based DSNs (BTS-DSNs), and other DSNs. In the remaining part of this section, we will discuss each category of DSNs in every subsection respectively. We additionally provide a table in our website[4)] to summarize software projects used to construct DSNs in our paper collection.

3.1 Project participation based DSNs

A project participation based DSN (PP-DSN) is constructed using the following rule: if two developers have both participated in the same project, there is a link between them in the DSN.

Following the above rule, every pair in the same project is connected, and the whole network contains all the developers (who have ever participated in at least one project) in SourceForge.Net. This is a reasonable DSN construction method for SourceForge.Net. In SourceForge.Net, most projects are small and involve only a few of participants. It is expected that two developers working in the same project have common interests and skills. As a result, the cooperation relationships of developers are used to construct DSNs. Madey et al. [6] firstly used project participation connections to construct and study DSNs in 2002. Subsequently, several researchers employed this type of DSN construction method [7–10].

The PP-DSNs is based on data from a platform with a lot of projects, such as SourceForge.Net. SourceForge.Net is a web-based project support site, providing project management tools, bug tracking,

3) This paper does not distinguish the roles of developers.
4) http://software.nju.edu.cn/iSE/DSN.

mailing list service, discussion forums, and version control software. Open-source software projects can be managed on this site, and developers also can choose their favorite projects to participate in. This kind of connection is very close, because the scales of most projects are usually small (less than 10 developers) and their tasks are very concentrated. Besides SourceForge.Net, the PP-DSNs can also be created based on other platform like Github [11].

The main limitation of PP-DSN construction method is that it is too coarse in some cases. It cannot be used to investigate on a specific project. Developers in a PP-DSN are even not in one project, but include all of users in the website of management platform. It considers coarse-grained relationships of project participation in DSN construction. Some detailed relationships within each project are ignored, along with the temporal relationships in this type of DSN construction. If a project has proceeded for many years, the method still links a current developer with someone who has quit the project 10 years ago. Obviously, they may have no connection in software development in this case.

3.2 Version control system based DSNs

A version control system based DSN (VCS-DSN) is constructed using the following rule: if two developers have both committed the same file or module, there is a link between them in the DSN.

Lopez-Fernandez et al. [12] firstly used VCS-DSNs in 2004. They constructed DSNs as weighted undirected networks. Two developers who have contributed to at least one common module are linked, and the weight of the corresponding edge is the number of commits performed by both developers to all the common modules.

In general, the change logs of software could be used to construct DSNs [12–23]. This type of DSN construction method focuses on activities in one project. The most direct way is to collect histories recorded in Version Control System (VCS), namely the code committing activities. The source code may be the most important outcome of software development. The source code could be managed by version control systems, such as CVS [12], SVN [19], Git [20]. When developers complete programming or fix some bugs, they will commit their current files and the VCS will update the source code repository according to the modified version. The change log records the developers committing files. It means that developers are connected by the files which they have modified.

The first characteristic of VCS-DSNs is that they are undirected. These undirected links of developers are from their common features, such as working on the identical project, modifying the identical software entity, etc. We also could introduce more information, such as temporal relationships of developers, to construct directed DSNs. This will be studied in our future work.

Different source code relationships can be used to construct VCS-DSNs. An appropriate granularity is important to VCS-DSNs. It should be determined by the size of a file, the frequency of commits, and the responsibilities of developers. It is better to choose coarse granularities such as module or class (e.g., [12]) in these cases as many files are too small, with only a few of commits performed on a file every day, and two developers seldom modify the same file. In general, VCS-DSNs with module relationships will be more dense than VCS-DSNs with file relationships [14]. The link granularity of VCS-DSNs should be decided according to the specific situation with real project data.

VCS-DSNs can be either a weighted graph or an unweighted graph, because the strengths of developer links may not be always the same. The weights of edges in DSNs indicate the strengths of developer relationships. The weights can be calculated by several ways. The weight can represent the number of common files that two developers have both committed [20]. The weight can be defined as the number of commits performed by both developers to all common modules [12]. The weight can be considered as a kind of metric and researchers could compute the weight by complex formula [19,21,22]. Also, the weights can be ignored for constructing an unweighted DSN. This can simplify the analyses and reduce the cost.

A critical factor of VCS-DSNs is time duration. A DSN can be constructed based on the data from one month, two months, or longer. The simplest way is to use all the data. The corresponding record is taken into account no matter when the committing event happens. In this way, a developer joining the project now may be connected in a DSN to another developer who left the project 3 years ago. To avoid

this kind of unreasonable situation, VCS-DSNs can be constructed using data recorded during limited time duration [22]. Meenely et al. [14] constructed VCS-DSNs with the software process data during every development cycle, which produces a released version. If a specific time is selected, it needs careful consideration to determine the duration length. One month is chosen in some cases but there are no detailed explanation, probably out of experience and convenience in many papers [22]. Pohl and Diehl used time sliding window to conduct dynamic analysis of evolution of DSNs [15].

In summary, VCS-DSNs can fairly capture the coordination relationships among developers. The assumption of VCS-DSNs is that two developers are similar if they modify common source code. The source code modification may be the most important action of developers. VCS can provide structural information. However, this modification link of developers is not a straightforward "social" relationship. This relationship is established in source code, not immediate communication.

3.3 Email archives based DSNs

Communication is essential for developers in software development life-cycle. Communication can be performed in many ways, such as face-to-face, telephone, and meetings. These ways are highly efficient and effective, but they are not satisfactory for geographically distributed software teams. The popularity of open source software also accelerates the globalization of software development. Nowadays, communication via the Internet has gone popular in such a way that all the histories of communication could be recorded and made accessible. Email is a popular communication method utilized by software teams, because it is straight forward and the server is easy to deploy. Therefore, email archives based DSNs (EA-DSNs) was born. The construction rule of EA-DSNs is: if two developers communicate via email, there is a link between them in the DSN.

It is a straightforward way of constructing EA-DSNs because people around the world often communicate via email [24–32]. Software teams often use public email lists, which everybody can see and participate in. Anyone sends an email to this list and then all the people in the list can receive it. Anyone interested in this topic can reply and others can also see the reply email. This kind of communication form works in a similar way as a forum. Project leaders in a project can release announcements, developers can talk about technical problems, and users can report bugs.

There are some challenges to construct EA-DSNs: (1) emails are textual messages written in natural languages; (2) a single developer may use multiple email addresses; (3) email addresses may be difficult to be mapped to usernames in the VCS; (4) some email messages may be noise. Bird et al. [26] proposed many heuristics to overcome the above challenges. They classified the topics of emails into either process topics (e.g., general announcements) or product topics (e.g., technical discussions for bug fixing). Their study shows that DSNs constructed only by emails about product topics are more related to development activities.

There are some communication links in the email list of a development team. Someone sends an email to this list, and then anyone in this list can see it. Once if another one replies this email, a social link appears. It is natural to use communication relations to construct EA-DSNs in the following cases.

(1) If a developer A responds to another developer B, a link $B \to A$ could be constructed [26]. In this case, the link $B \to A$ is directed and denotes that A receives and understands B's message. In a public mailing list, if B sends an email then A replies to B, the link $B \to A$ is constructed. This kind of DSNs is to describe the situation of information transmission.

(2) An email sent to the public mailing list may achieve several developers' replies. According to Case 1, each of these emails can also be used to construct a link from the sender to the replier. In this case, all the developers who participate in the same topic can be considered to have common interests, so each pair of them can be linked in a DSN [31]. This kind of DSNs is an undirected graph.

(3) Some work does not mention public mailing lists clearly [27]. They may use internal emails in the general way. So in these papers, a link from developer A to B in DSNs may be constructed when A sends an email to B.

3.4 Bug tracking system based DSNs

A bug tracking system (i.e., defect/issue/problem tracking system) [33] is a repository to manage bug reports within the life-cycle of software. In bug tracking systems, communication is performed by comments, so DSNs can be constructed by relationships of comments [33–38]. The first rule to construct bug tracking system based DSNs (BTS-DSNs) is: if two developers both comment on a common bug report, there is a link between them in the DSN. In a bug tracking system, bug reports can be created, assigned, modified, commented and closed. Bug reports are usually structured with many predefined fields, e.g., title, component, severity, version and so on, to facilitate bug fixing. In addition, some so-called bug reports are actually requests of new features or enhancements. In this sense, the bug tracking system becomes the modification request system, where developers manage not only bug reports but also other modification requests. Since developers' communication and coordination are recorded structurally in either bug tracking systems or modification request systems, DSNs can be mined and constructed from bug repositories in various ways.

This kind of relationship is based on common interests and skills of developers. Here, comments can be replaced by general participation, because bug reports can be not only commented, but also modified and assigned to, etc. It arises to be another DSN construction pathway with comments of bug reports [24,39]. The method mines directed links by concerning reply-to relationships in comments as the second rule of BTS-DSNs: if developer A replies to developer B's comment of a bug report, there is a link from A to B in the DSN. This kind of relationship is much closer than others before. Two developers of a link comment a common bug report, and communicate with each other directly. It should be noted that the comments are always considered as a reply to its previous comment in bug tracking systems. It is not clearly labeled as replies to which one of previous comments. However, the comments in a bug report discuss about the same topic and a comment is often submitted to reply to the latest opinion. This kind of DSNs is constructed as a directed graph using comment information of bug reports.

The above two ways both construct one DSN based on all of data. A BTS-DSN can be constructed based on comments for some specific files in the third way [40]: for each file, if a bug report is related to this file, then add the relationships associated with the comments of this bug report to the DSN for this file. This rule builds a BTS-DSN for each file, in order to compute a certain file-based metric. It uses the relations between version control systems and bug tracking systems. It is difficult for many projects, because these two types of systems are often separated (unless those projects which can make the links between the two types of systems straightforward), so that no information can be located to a file with a bug report directly.

Besides comments or general participation, other specific actions in bug tracking systems attract researchers' attention to construct DSNs. For example, bug reports can be reassigned. When an assigned developer cannot fix the current bug report, it can be reassigned to another developer. This phenomenon is very common in bug tracking systems. Many bug reports are reassigned for many times until fixed. BTS-DSNs can also be constructed to capture this phenomenon as follows [41,42]: if a bug report is reassigned to developer A, and its previous assignee is developer B, there is a link from B to A in a DSN. The tossing process of each bug report is mapped into a path passing through developers. All these paths make up the whole DSN, a directed graph. Bug tossing graphs can be used to help triagers in software teams find the proper bug fixer when the current assignee of a bug report cannot fix it.

There exist some other methods to construct BTS-DSNs. Zhang and Lee [43] proposed a special rule to construct BTS-DSNs as follows: in a bug report, if developer A is assigned to fix this bug and developer B comments this bug report, there is a link from A to B in the DSN. This rule connects a directed link from the assignee to every commenter of a bug report. In this way, a model is built to predict the most appropriate bug fixer. The assignee and the commenters of a bug report are both related with it, but the assignee has much closer relationship. This is because the commenters may be just interested in this bug report, while the assignee needs to have relevant skills to fix it. Zanetti et al. [44,45] used another two rules to construct BTS-DSNs: (1) if developer A adds developer B to the CC list of a bug report, there is a link from A to B in the DSN; (2) If developer A assigns developer B a bug report, there is a

link from A to B in the DSN. The CC interaction means that A is aware of B and A knows B's interests. The assignment interaction means that A and B take up different roles. Wu et al. [46] proposed the following BTS-DSN construction way: for a newly coming bug report, a DSN is constructed such that nodes are all the potential assignees and every pair of nodes is linked with the weight of the similarity between the two developers. The motivation is to effectively assign bug reports. When a new bug report comes, the bug reports similar with the new one are found by their features. Then the developers who have participated in these bug reports are defined as the nodes of a DSN. Every pair of them is linked in this DSN, and the weight of a link is the similarity between the two developers. This type of DSN is constructed every time when there is a new bug reported. This kind of construction is very special and appropriate particularly for its application scenario.

3.5 Other DSNs

In addition to the above categories, some DSN construction methods are specific and not commonly used in a certain repository. It is natural that DSNs can be constructed according to the predefined organizational structure in an enterprise [47,48]. This method is only applied for traditional commercial software projects, because their organizational structures are defined carefully and clearly. The related data needs to be collected from either documents or interviews. However, this method is difficult to be employed in open source software projects, since no predefined organizational structure exists. DSNs can be constructed with the recommendation grades that a developer gives to another one. Wagstrom et al. [25] conducted a study based on a website where developers can grade one another. Lim and Bentley [49] conducted the study by interviewing stakeholders, and asking developers to grade others. If A can give a grade to B, it means A understands B's skills, so a link from A to B is added with the grade as its weight. DSNs can be constructed according to their communication during the process of code integration build [50]. Communication information collected to construct DSNs is recorded on the particular code integration build platform. DSNs can be constructed from the transformation of technical networks [51,52]. These papers analyze code's caller-callee dependencies to achieve a technical network, and transform it to a DSN, which indicates their technical dependencies. DSNs can also be constructed with the cross links in developers' blogs [25]. This method is also used in a special scenario that every developer has a personal blog main page. Those developers often write technical articles, and can read and refer to articles in other developers' blogs. Such cross links are picked out to construct DSNs.

In addition to the single types of DSNs above, there exist some hybrid DSNs which are not pure DSNs. These nodes in hybrid networks include not only developers but also some other entities. Developers are still related to each other, although not directly, in these hybrid DSNs. The most prevalent kind of hybrid networks is the socio-technical network [53–57]. These hybrid DSNs contain both software modules and developers. There can be three kinds of links in a socio-technical network: (1) module to module, based on software dependency, (2) developer to developer, based on developers' cooperation, (3) developer to module, based on the ownership. Surian et al. [58] constructed a network made up of developers, projects and project features. Similar to PP-DSNs, the source of experimental data is SourceForge.Net. There are two kinds of links in the hybrid DSNs: (1) developer to project, which indicates what projects a developer has participated in, (2) project to project feature, which indicates what features a project has. Some other software entities, e.g. requirements, bug reports, test cases, etc. may be also counted to construct hybrid DSNs [56]. A lot of attempts are needed to see if any network analysis method is useful for these networks in a particular scenario in the future.

4 Analysis of developer social networks

As a specific social network, DSNs have been studied for a relatively short time. Some simple methods from the area of Social Network Analysis (SNA) have been introduced in DSNs. Based on SNA, the properties of social networks can be discovered. Some pertinent conclusions and predictions are made that explain a series of phenomena in software engineering and improve software development activities.

We review the general SNA methods used in DSNs in Subsections 4.1–4.7. The bridge to the gap between SNA methods and software engineering is discussed in Subsection 4.8.

4.1 SNA metrics

A social network has the small world and scale free properties. Many SNA metrics have been proposed to capture the properties of the whole network and a single node [40,53,54]. We review three types of SNA metrics: centrality, global metrics and other metrics.

Centrality is designed to measure how central a node locates in the network. Different criteria have been proposed to decide the value of a node's centrality [12,14,27,35,40,45,46,53,54]. With the first three metrics in the following list, the centrality of a developer can be determined reasonably in most situations. The centrality of a node equals to the importance of a developer basically in DSNs.

(1) Degree Centrality is defined as the number of neighbors of a node [14,45,50,53]. If the DSN is directed, there will be in-degree and out-degree. Degree centrality is the simplest way for measuring the centrality of a node. If a developer has many neighbors, he/she is apparently popular. However, this metric is just local to the node itself, without considering other parts of the whole network.

(2) Closeness Centrality is defined as the number of steps required to go from the current node to all the other nodes [14,45,53,54]. Closeness centrality is also called connectivity or distance centrality. It measures how closely a node is located to other nodes. If the distance is low, the location of the node is central in the network.

(3) Betweenness Centrality is defined as the number of shortest paths between pairs of other nodes that run through the node [14,45,50,53]. Betweenness centrality is used to measure brokerage or information flow. More shortest paths run through a node, more important this node is.

(4) Bonacich Power is defined as a variant of degree centrality [53,54]. It merges the degree of the current node and the degree of its neighbors. A node is considered central if it is connected to the nodes that have connections to many other nodes.

(5) Reachability is defined as a variant of closeness centrality [53,54]. It measures the connectivity of a network. It denotes the portion of other nodes in the network starting from this node can be reached with a particular number of steps.

(6) Barycenter Centrality is calculated based on the sum of the lengths of all the shortest paths that pass through this node [40]. More central nodes will have smaller overall shortest paths.

(7) Eigenvector Centrality is defined as a sophisticated centrality measure with the feedback feature (e.g., PageRank) [45,54]. Nodes connected to highly central nodes increase their own centrality recursively.

Global metrics are designed to investigate the network as a whole. The most simple global metric may be the size of network, i.e. the number of nodes or edges. Diameter is defined as the biggest value of all possible shortest paths between every pair of nodes in the network [7,40]. Higher diameter indicates that it takes longer time that information travels the whole network. Density is defined as the proportion of edges in a network to the total number of all the possible edges [40,50]. Higher density indicates that the nodes in the network have tighter connections with one another.

There exist some other metrics. Clustering Coefficient is calculated as the probability that any two neighbors of the current node are connected [7,12,29,40,44,45]. This metric measures the local connectivity density. Group Degree Centrality Index is defined as a measure of how central a network is [44]. The network with the star topology has a higher value for this metric. Characteristic Path Length [44] is defined as the average number of edges in the shortest path between nodes. Smaller value means that information can spread more easily. K-coreness [45] is defined in this way: if a node has a degree k after removing all other nodes with degree up to $k-1$, this node belongs to a given shell k. Number of 2-paths [29] is defined as the number of 2-paths going through a node. It is a measure of local social status. Brokerage is defined as the number of pairs of nodes that are connected only by the current node [30,54]. Nagappan et al. [47] proposed a set of special metrics, i.e. organizational metrics. Those metrics are appropriate for hierarchically organized structures.

4.2 Roles

The main purpose of SNA is to evaluate the importance of a node. The above mentioned centrality metrics can assess quantitatively. The software engineering tasks always need to decide whether a developer is an expert or plays some particular role. For example, Ricca and Marchetto [59] found that "heroes" are common in open source software projects and they exclusively manage a conspicuous proportion of files and code. We can use SNA to figure out the role that a developer plays in a software project.

Developer Ranking. Many researchers use SNA to find leaders or key developers in DSNs. A ranking method is used to estimate which developer is more important than others. Centrality is the most popular way to measuring the importance of developers. PageRank is the algorithm used by Google search [60] and it can be also used for developer ranking [46]. The importance of a node is also evaluated by the probability of "visiting" the node when a user joins the network [32]. This method considers both the current node and its neighbors. HITS score [32] is a score method first proposed in the hyperlinked environment. It combines both inlinks and outlinks of a node.

Some other efforts also concern developer ranking or prioritization [42,43,56]. These ideas are either borrowed from the general social network research or improved based on PageRank. In addition, these methods are closely related with specific scenarios, e.g., bug triage or bug severity prediction. The developers can be divided into the following two groups [13,21]: core developers, who lie in the center position in DSNs; periphery/associate developers, who lie around core developers in DSNs. We can also determine whether a developer belongs to core or periphery by either visualization [21] or clustering [13]. It should be noted that core developers may have different definitions in various papers.

Information Flow. Information can travel along the edges in DSNs, because DSNs depict communication and coordination relationships among developers. Developers playing crucial roles in information flow need to be identified. The simplest method is to compute degree centrality. Hub is a developer with a high degree [14]. A hub developer is relatively crucial with heavy burden in the project. However, the threshold that decides a hub developer varies with projects. Other than important nodes, one can also find important edges. Bridge is an edge whose endpoints will be completely disconnected if it is removed from the network [40]. Bridges are important channels for information transition. Two developers or even two communities cannot interact with each other when a bridge is broken.

According to their roles in information flow, developers can be categorized into three kinds in a directed DSN [30]: coordinator, a developer who lies in the path between nodes within the same cluster; gatekeeper, a developer who lies in the path between nodes of other clusters and nodes within the same cluster, as a filter for the incoming information; representative, a developer who lies in the path between nodes within the same cluster and nodes of other clusters, as a proxy for the outgoing information. To identify the three roles, one needs to find clusters of DSNs in advance. A social network is a kind of complex network with the small world and scale free properties, and DSNs contain some community structures (or clusters). In a cluster, a coordinator is an information broker within its cluster. A gatekeeper or representative is an information broker across clusters. Damian et al. [48] found that brokers (namely gatekeepers or representatives) are usually boundary spanners who have many communication links across domains. Actually, betweenness is the most popular method to identify brokers [26]. In addition, there are also information brokers during the requirement phase [61].

When connections are imbalanced, there will be an interesting phenomenon. Structural holes [50,54] are gaps in the social network to indicate the diversity of information flow. For example, A has a neighbor B, but B does not have other neighbors. Then the absence of an edge between B and A's other neighbors represents structural holes. This measure is concerned with the degree to which there are missing links in between nodes and with the notion of redundancy in networks. Researchers induce this concept into software engineering as a measure to be used in prediction models.

4.3 Community structures

The community structure or modularity is a key characteristic of a social network. Many researchers have evaluated whether it still holds good for DSNs and mine corresponding communities. Surian et al. [9]

Table 1 Variable with power law distribution in DSNs

Variable	Papers
Degree of nodes	Refs. [7,12]
Connectivity of nodes	Ref. [9]
Out-degree of nodes	Ref. [26]
In-degree of nodes	Ref. [26]
Number of nodes of communities	Ref. [9]
Number of messages sent by developers	Ref. [26]
Number of messages read by developers	Ref. [26]
Number of projects with n developers	Refs. [6,7]
Number of developers on n projects	Ref. [6]
Stakeholder involvement	Ref. [26]

studied the connectivity of DSNs. As a result, the DSN in their work is not a connected one, but is made of many disjoint connected components or clusters. In this case, every connected component is considered as a community. Madey et al. [6] also identified clusters by connected groups of developers. Both these methods construct PP-DSNs. There exist many connected clusters in their works. Zanetti et al. [45] used a special DSN construction method based on "CC and ASSIGN relationships in bug reports". In such a case, a DSN is not connected either, but there is the largest connected component containing most of nodes.

The reason why a DSN is not connected but contains many disjoint connected components lies in that the DSN construction method makes the DSN too coarse. The DSNs are usually connected if DSN construction uses email archives, version control systems, or bug report comments [20,28,39]. The visualization of the DSN is to show the DSN (a VCS-DSN) contains several small strongly connected groups [21]. Developers in the same community usually belong to the same predefined team. In addition, the organizational structure can be revealed. For example, one can know who plays the role of a broker.

Community structures are identified by community detection algorithms for large scale DSNs [28,36,38]. DSNs are constructed from "email communication" [28] and "common bug report comments" [36,38], respectively. To detect community structures, a variant of Newman and Girvan community detection algorithm is used in [28,36,38]. In this algorithm, a measure called "modularity" is used to quantify the strength of a community structure. If connections are dense within each community and coarse across communities, the modularity is high. These DSNs contain clear community structures, since the modularity of each DSN is high [28,36,38]. The communities detected can often indicate the organization of the project in the real world. For example, developers in one community probably come from the same product team. In addition, Damian et al. [62] described a decision tree method enabling practitioners to uncover latent social communities in software development.

4.4 Properties

Social networks are also small world and scale free networks [63], and they have three common properties.

(1) The average length of the path between any two nodes is short relative to the size of network. For example, six degrees of separation is a popular theory in social network, i.e., anyone can reach another one within six steps [64].

(2) There exist clearly community structures in the network. In another word, the clustering coefficient and the modularity measure are both adequately high. Some researchers find that communities can be mined from DSNs [9,28,36,45].

(3) The distribution of degree in the network should follow the power law distribution [65].

Many efforts verify whether the power law distribution can be found in DSNs, not only for degree, but also many other variables [6,7,9,12,26,36]. We summarized them as shown in Table 1.

The property of the power law distribution is found on DSNs constructed by different methods, including PP-DSNs [6,7,9], EA-DSNs [26], VCS-DSNs [12], and BTS-DSNs [36]. The characteristic of the power law distribution is that the diagram looks like a straight line, when depicted in log-log scale. However, there are also some exceptions which do not satisfy the power law distribution. For example, the distribution of degree does not follow the power law in [36]. Similarly, the number of edges and node degrees of the core cluster in [9] and clustering coefficient of modules in [12] do not follow the power law distribution. It should be noted that there is a common feature between these exceptions and the power law distribution. Using the typical example, i.e. the degree of nodes as the variable, there are very few nodes with high degree and far more nodes with low degree, no matter whether it follows the power law distribution. This phenomenon indicates that there are significant leaders in a DSN.

The small world property means that nodes are closely connected in networks. A typical example is six degrees of separation, which means that all developers are connected to every other developer within at most 6 hops. Researchers can use the average path length to study this phenomenon. The average path length is the average of the geodesic path (i.e. the shortest path) between every two node. Each of the average path lengths is no more than 6 in the studies [7,9,36,38]. Hong et al. [36] showed the distribution of the length of geodesic paths, and their diagram indicates that the length of most geodesic paths is about 3. Xu et al. [7] used the diameter, i.e. the largest geodesic path, to demonstrate the small world phenomenon. In summary, studies in the literature have shown that DSNs could considerably satisfy the small world property.

4.5 Evolution

Evolution is an important topic in the research area of general social networks. However, it has not been able to attract much attention in DSNs. Hong et al. [36] discussed the evolution of different properties, e.g., the power law distribution, degree of separation, modularity, and community size. Kumar and Gupta studied the evolution of different kinds of measures, e.g., the number of contributors, clique size, clustering coefficient, average degree, average path length, and average distance [38]. Cataldo and Herbsleb demonstrated how DSNs in geographically distributed projects evolve [2]. Lim and Bentley explored evolving relationships between social networks and stakeholder involvement in software projects [49]. Sharma and Kaulgud applied social network analysis techniques to investigate team evolution in a project's testing phase [66]. Research on the evolution of DSNs can help us understand the changes of software project teams and software process.

4.6 Patterns

As DSNs are represented graphically, traditional graph analysis techniques can be conducted on DSNs, e.g., graph mining (called collaboration pattern mining in DSNs). Jermakovics et al. [21] investigated a kind of patterns named topological patterns, i.e. organizational structures in DSNs. There exists a direct relationship between collaboration patterns and the outcome of the software development [9]. In particular, a study indicated that collaboration among developers is an important factor that influences the issue resolution time [67]. It is also necessary to detect better collaboration patterns for software managers for reference. Surian et al. [9] proposed a study to mine collaboration patterns from DSNs based on historical collaborative information. The combination of graph mining and graph matching is used to get the frequent sub-graphs. Then, the top 30 topological collaboration patterns can be extracted. After analyzing those collaboration patterns, the authors reveal that the frequent patterns are of small sizes. However, those frequent collaboration patterns cannot able to predict project outcomes. Surian et al. [57] further proposed an effective approach to extract discriminative collaboration patterns in DSNs. The discriminative collaboration patterns could distinguish successful projects and failed ones by a high accuracy. For mining discriminative patterns in DSNs with multiple labels, a translation process is used to map this problem to mining simple graph patterns.

4.7 Visualization

Visualization is important for social networks. If a DSN is visualized in an appropriate way, one can derive many conclusions easily from the graph. Some papers conduct their analysis based on visualization of DSNs [21,24,51,55,56,68–70]. For example, one can find out core developers, information brokers, and community structures. In [21], Jermakovics et al. used visualization as a primary method of analysis. They drew DSNs of several open source projects, and showed many analysis results visually, e.g., team structures and central members, different roles in the organization, and small communities. Begel et al. [56] designed a framework called Codebook, to present and connect people with their work artifacts. With this tool, developers can coordinate with one another more conveniently and efficiently. Sarma et al. [55] developed a socio-technical dependency browser called Tesseract, in order to show hybrid DSNs and help software engineers understand their software teams.

4.8 Bridging the gap between DSNs and SE

It is important to combine SNA methods and the features of software development. In this subsection, we summarize the research works bridging the gap between DSNs and software development. Meneely and Williams validated whether SNA metrics could capture real situations [20]. They interviewed developers in real projects, and found that the results of SNA are in accordance with developers' perception. To be specific, edges in a DSN can indicate that two developers are collaborating; the distance between two developers in a DSN can stand for their degree of separation; the high centrality of a developer indicates that he/she is an expert in the project.

Many researchers investigate the relationships between DSNs and software development activities with some hypotheses.

(1) Developers within the same community are more likely to collaborate directly [31].

(2) The average directory distance between files committed to by developers in the same community is less than the similar sized group of developers drawn from different communities [28].

(3) The number of messages sent by a developer has strong relationships with the degree of this developer, and the number of source changes this developer makes [26].

(4) SNA metrics such as degree and betweenness indicate developers who actually commit changes [26].

(5) Developers are more likely to play the role of gatekeepers or brokers than non-developers in the complete email social network [26].

(6) The developers within the same module communicate with each other closely and frequently [28].

(7) The more modules a developer contributes, the more communication he/she will have with each other [31].

(8) There is a significant relationship between network centrality and coordination (A text-mining application is developed to measure the coordinative activity of each developer using information extracted from email datasets.) [27].

SNA metrics and other observations in DSNs are related to coordination in software development. Besides activities such as committing changes, results analyzed on DSNs also have relationships with other features in a project. For example, centrality is related to the size of project [24]; properties of DSNs are related to the type of developers contained in DSNs [7]. The more useful analysis of DSNs is to predict the future of projects. In [10], Antwerp and Madey found that relationships which have already existed before among developers are correlated with the success of open source projects. A novel social measure based on DSNs is proposed by Zhou and Mockus in [37], and verified to have relationships with the future of participants, e.g., whether a participant will become a long term contributor. More papers (e.g., [16]) use SNA metrics in bug prediction.

5 Applications of developer social networks

DSNs have been widely used in some applications associated with software development. As to the functional point of view, these applications can be roughly divided into two groups, i.e., the Prediction

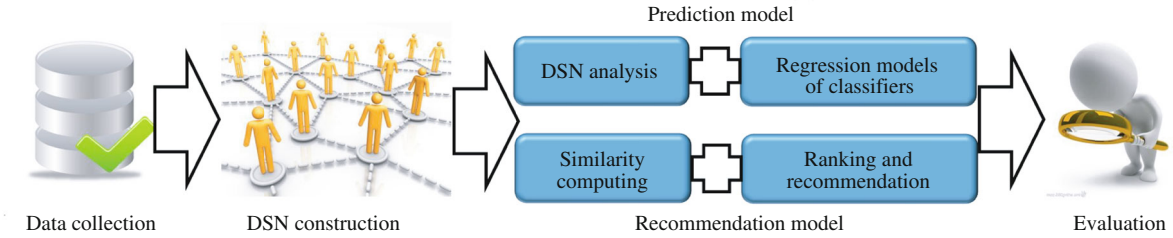

Figure 2 Procedures of two groups of applications.

Model based applications and the Recommendation Model based applications (see Figure 2). Both groups include data collection, DSN construction, and evaluation. The key difference between these two groups is that, the Prediction Model based applications need to construct either regression models or classifiers, while the Recommendation Model based applications compute and rank the similarity of developers for recommendation.

5.1 Prediction model based applications

In the following part, we first describe the procedure of the Prediction Model based applications. As shown in Figure 2, the Prediction Model based applications usually consist of 5 steps.

5.1.1 *Data collection*

To construct a DSN, relevant information during software development is collected, including the communication between developers, the contributions of developers to components, and the dependencies among components (e.g., modules, files, packages, functions, binaries, pieces, etc). More specifically, the communication among developers could be extracted from email archives [30,56,71]. The contributions of developers include comments [33,36,46,50], commits [14,42,53,56,72], bug fixing [41,42], etc. The dependencies among components include the calls of functions, class inheritance or coupling, and system dependencies [54,73], etc.

The collected information could be stored in different software artifacts in distinct projects. For commercial software projects (e.g., Window Vista [47,53,54,74], Hibernate-ORM [23,52], and IBM Rational Team Concert (RTC) [57,72]), the related information is stored in Jazz [71], people management software [47,56], etc. In contrast, the related information for open source software projects (e.g., Eclipse [36,38,41,42,45,54,73,75], Mozilla [33,36,41,42,45,46,75], SourceForge [9,57,58], and Netbeans [45,73]) is stored in Bugzilla [39,41], Trac [17], and Git repository [52,76], etc. Meanwhile, in some studies, some specific tools are developed to extract such information, e.g., MaX, Understand from SciTools [54], Dependency Finder [73], and CodeViz [42], etc.

5.1.2 *DSN construction*

In a DSN, the nodes represent either developers or components, and the edges represent the connections of nodes, e.g., communication, contributions, and dependencies, etc. With respect to the information used, DSNs can be called as contribution networks [14,53], communication networks [30,71], collaboration networks [42,45,53,58], and dependency networks [54,77,78], etc.

5.1.3 *DSN analysis*

The metrics used to measure DSNs can be divided into two types, i.e., complexity metrics and network metrics. The primary differences lie in that complexity metrics mainly focus on single elements, while network metrics focus on the relationship between nodes in networks. Generally, complexity metrics include the number of authors, number of commits [53], number of lines of added/changed code [14], number of minor contributors [74], and density [50,71], etc. Some typical network metrics include the

degree centrality [77], closeness centrality, betweenness centrality [14,53], eigenvector centrality [45,54], and PageRank [46], etc. In the literature, many tools have been designed to compute these metrics, e.g. Java Universal Network/Graph library (JUNG) [40,71,73], UCINET [54], SPSS [53], ORA [67], and B_LIN [58], etc. Meanwhile, the method of Principle Component Analysis (PCA) has been widely used to select the nonlinear combinations of these metrics.

5.1.4 *Regression models or classifiers*

In order to predict, one first needs to build either regression models or classifiers on training data, and then employs either regression models or classifiers. According to distinct requirements, different regression models or classifiers can be chosen, including negative binomial regression, Poisson regression, logistic regression [14,47,53], Bayesian classifier [50,71], and Naive Bayes [39–41], etc. To test how effective the regression models are when predicting, Pearson and Spearman correlation coefficients between the predicted values and the ground truth are used. For classifiers, some tools could be used, e.g., Weka [39–41], and Support Vector Machine (SVM) [39,45,57], etc.

5.1.5 *Evaluation*

The metrics of precision and recall are widely used by most studies to evaluate the results of classification and prediction [46,47,50]. In addition, other metrics are also used, such as AUC (area under the ROC curve) [54,57], Accuracy [39,41,57,58,75], and F-score [54,73].

5.1.6 *Similarity computing, ranking and recommendation*

Different with the Prediction Model based applications, in the Recommendation Model based applications, one computes and ranks the similarities of developers for recommendation. There are three types of similarities; the similarities among developers, the similarities among projects, and the similarities among developers and projects. Those similarity information can support developer recommendation and bug triage, or just be used to construct DSNs (considering the similarity as the weight of an edge). As general recommendation system techniques are often combined with general social network analysis, similar studies have been performed in the area of DSNs. The commonly used methods for calculating the similarities include collaborative filtering [19,22,41,58,79], the full-text search algorithm of SQL Server [56], and the Random Walk with Restart (RWR) [58], etc.

Since the information for constructing DSNs is primarily collected in software maintenance, most applications of DSNs are conducted within software maintenance. From Subsection 5.2 to Subsection 5.5, we detail typical applications of DSNs in software maintenance. In Subsection 5.6, we introduce some applications out of software maintenance.

5.2 Integration build quality prediction

This subsection mainly introduces the studies for integration build quality prediction. These studies share the common characteristics as follows. First, they all belong to the Prediction Model based applications. Second, all of these studies are conducted on commercial software projects. Finally, in these studies, some of five steps in the Prediction Model based applications are completed by specific tools, e.g. Java Universal Network/Graph (JUNG) [71], and Bayesian classifier [50,71], etc.

Wolf et al. [71] employed task-based potential relations in DSNs to predict the outcome of integration build. The JUNG framework is used to construct and visualize multiple nodes of DSNs on the Jazz project. The involved data are extracted from multiple forms of communication, such as the comments of developers to bug reports, emails among developers, etc. Wolf et al. [50] also studied the relationship between the team communication structure and the quality of software integration in DSNs. After a case study of IBM's Jazz project, they suggested that the combination of some communication structure metrics can be used to identify whether the integration build would fail. For detecting the relationship between socio-technical congruence in DSNs and build-success probability, Kwan et al. [72] proposed a method using both weighted congruence measures and unweighted congruence measures. In their study,

this method is proved to be useful to find the lack of coordination. After performing a case study in the RTC project, Kwan et al. revealed that the socio-technical congruence is not simply correlated with build success (If the build type is a continuous build, then increasing the congruence leads to an increase in the build success probability. If the build type is an integration build, then increasing congruence actually leads to a decrease in the build success probability.).

5.3 Defect prediction

Defect prediction is a process, in which the historical data are used to identify the components that are most likely to fail afterwards [80]. The prediction of component defects can support managers to adjust their behaviors and reduce testing costs. Nagappan et al. [81] and Schroter et al. [82] issued the failure-prone components prediction based on complexity metrics. The studies for defect prediction based on DSNs can be divided into two categories: the predictive studies based on DSNs extracted from churn information and dependency relationships [14,53,54,73,83], and the predictive studies based on DSNs extracted from quantitative analysis [23,33,42,47,52,74,84–87]. Moreover, these studies also involve the balance between the probability of detection and the probability of false alarms [40].

5.3.1 *Defect prediction based on DSNs extracted from churn information and dependency relationships*

In DSNs, the churn information can reflect the contributions of developers to components [14,53], e.g., comments, commits, updates, etc. The dependency relationships between components include class inheritance or coupling, function dependencies, etc. [54,73] The data for constructing DSNs are taken from revision control repositories, bug tracking systems, mailing lists, etc. Next, we present the details of these studies.

Pinzger et al. [53] and Meneely et al. [14] constructed DSNs based on the churn information to predict component defects in two different projects respectively. Specifically, Pinzger et al. [53] built a contribution network with two types of nodes on the Vista project. In this network, two sets of nodes represent developers and binaries respectively. The edges represent the contributions of developers to binaries. The weights of edges represent the number of commits performed by a developer for a binary. Meneely et al. [14] built a DSN with one single type of nodes using the data of Nortel Networks. In this network, the nodes represent developers. An unweighted edge between two developers denotes that both developers have collaborated on at least one file during the same release. For quantitative analysis of the centrality of nodes in DSNs, some network metrics are used in both studies, e.g., degree centrality, closeness centrality, and betweenness centrality, etc. Moreover, the complexity metrics are also used in both studies, e.g., the number of authors, number of commits, and number of lines of added or changed code, etc.

The dependency relationship also proves to be useful for predicting defects. In the work of Bird et al. [54], the dependency graph and churn information are combined to generate a new type of DSN, namely hybrid socio-technical networks. The authors reveal that hybrid networks achieve higher accuracy than the networks using either dependency relationships or churn information only. Hu and Wong [73] extended the work of Bird et al. [54], and suggested that the strength of relations among nodes can be used to predict the number of post-release defects. In their study, the strength of relations is measured with a topic model, namely the citation influence model. The results of an empirical study show its feasibility and high accuracy.

5.3.2 *Defect prediction based on DSNs extracted from quantitative analysis*

Researchers employ two kinds of quantitative analysis to construct DSNs, namely quantitative analysis of organization structures and quantitative analysis of ownership.

(a) Studies based on DSNs extracted from quantitative analysis of organization structures.

Brooks [5] stated that product quality is seriously affected by organization structures. Nagappan et al. [47] investigated the relationship between organization structures and software quality in Windows Vista. In their work, eight network metrics are proposed quantifying the organizational complexity in DSNs. The results of an empirical study show the efficiency of those organizational metrics for identifying

failure-prone binaries. Bird et al. [85] also revealed that the organizational differences are much stronger indicators of quality than geography in DSNs.

Moreover, the software quality is seriously affected by the organizational change and organization structure risks. Mockus [86] pointed out that the organizational changes can increase the probability of software defects. Bhattacharya et al. [42] also claimed that stable development teams can reduce the number of defects. In order to find the risk of organization structures, Sureka et al. [33] used a collaboration network derived from bug report repositories of Mozilla Firefox, to predict the risk and vulnerability in organization structures. Amrit et al. [88] proposed TEchnical Social Network Analysis (TESNA) methods, and develop tools to identify Socio-Technical Structure Clash (STSC) in a medium industrial sized company, namely eMaxx.

(b) Studies based on DSNs extracted from quantitative analysis of ownership.

Besides organization structures, software quality is also seriously affected by the experience of developers [89]. In general, one intuition is that a highly experienced developer could produce high quality software, and a developer lacking experience may produce low quality software. In the domain of DSNs, the experience of developers is usually measured by the ownership metric. For the empirical studies on Windows Vista and Windows 7, Bird et al. [74] demonstrated that the ownership metric shows a stronger relationship with pre-release failures than post-release failures. In addition, Bird et al. also revealed that there exists a strong positive relationship between the number of minor contributors and failures. Similarly, a pair of developers with low experience may also produce low quality software. For identifying the defects induced by developer pairs, Ell [52] and Simpson [23] conducted their respective studies on the Hibernate-ORM project. In both studies, the authors used the Failure Index (FI) to determine the failure-inducing possibility of developer pairs in DSNs.

Moreover, the failure-prone prediction aims to not only achieve better accuracy, but also reduce the probability of false alarms. Biçer et al. [40] proposed a set of metrics on issue repositories to achieve the best balance between the probability of detection and the probability of false alarms. The churn metric is compared as a benchmark on the empirical study about the developers' comments on the issues of IBM Rational Team Concert (RTC) and Drupal projects.

5.4 Bug triage

Bug triage aims to assign an appropriate developer for a new bug report [90,91]. To avoid the expensive cost of manual maintenance, Čubranić and Murphy [92] first proposed automatic bug triage. The traditional ways for bug triage are based on machine learning methods [92]. In recent years, some researchers facilitate bug triage with DSNs.

5.4.1 *Recommendation based on development experience*

Based on the bug reports and their comments in Mozilla Firefox, Wu et al. [46] proposed an approach, namely Developer Recommendation with k-nearest-neighbor search and EXpertise ranking (DREX), to recommend developers for new bugs. The expertise is measured by PageRank and the degree in a DSN. The recommended developers may share interests or potential knowledge on resolving the reported bug. Similar as the work by Wu et al. [46], Zhang et al. [43] proposed an automated method for developer recommendation based on the experience and fixing cost. Xia et al. [93] combined bug report analysis and developer analysis and proposed an accurate method for developer recommendation.

5.4.2 *Bug triage based on tossing graph*

Bug tossing is a phenomenon that a bug report is reassigned after its first assignment [75]. Tossing graphs prove to be effective in improving the accuracy of bug triage [41,75]. Jeong et al. [75] first proposed the tossing graph based approach to improve bug triage accuracy. A tossing graph is generated using the reassignment information in bug repositories. Based on the prediction list of Naive Bayes or Bayesian Networks, the developers with the large tossing probability are added to the new prediction list.

There exists some room for improvement in the work of Jeong et al. [75]. First, the equal-sized folds training method is inadequate for large fold sizes. Second, retired and inactive developers are not identified. Third, the tossing probability is insufficient for recommending the best developer. To deal with these problems, Bhattacharya et al. [41] proposed and verified several improvements on the Mozilla and Eclipse projects, including: (1) intra-fold updates rather than inter-fold updates to train the classifier, (2) the new tossing ranking function with multiple factors, (3) the developer expertise labeled on the edges, and (4) the developer activity labeled on the nodes in a tossing graph.

5.4.3 *Bug triage based on the importance of developers in DSNs*

Besides tossing graphs, quantitative analysis for the importance of developers in DSNs is used to improve the accuracy of bug triage. The importance of developers in DSNs is related with the developers' contributions and the effectiveness of their works. In this area, the contributions refer to code commits, comments on bug reports, etc.

Xuan et al. [39] first proposed the developer prioritization based on developer contributions in bug repositories. In their work, the developer contributions in DSNs are measured with the comments for bug reports. In the experiments of bug triage based on the developer prioritization, a prediction list is generated by an existing machine learning approach, and then the developers with similar probabilities in the prediction list are discriminated using the developer prioritization. Zanetti et al. [45] found that there exists a relationship between the centrality of bug reporters and the quality of bug reports. The centrality of bug reporters is quantitatively measured by nine topological metrics on DSNs. The relationship is used to identify valid bug reports for efficient bug triaging procedures.

5.5 Bug fixing

This subsection mainly introduces the studies of bug fixing based on DSNs, including the relationship analysis between the properties of DSNs and the effectiveness of bug fixing process [38], and cross-system bug fixing [30].

Kumar and Gupta [38] investigated the relationship between the properties of DSNs and the effectiveness of bug fixing process. After an empirical study, they concluded that more modular communication structures can improve the percentage and quality of bug fixing. Canfora et al. [30] found the phenomenon of Cross-System Bug Fixings (CSBFs). Canfora et al. extracted and constructed a DSN from the mailing lists and CVS, and then used various metrics to analyze the contributors involved in CSBFs. Their study shows that the contributors involved in CSBFs change more source code lines than other contributors.

5.6 DSN applications beyond software maintenance

Along with the progress of software development, a lot of information related to development can be collected, which can also be used outside of software maintenance.

In the phase of the requirement analysis, traditional methods may ignore some stakeholders and requirements. Meanwhile, traditional methods also ignore the diversity of stakeholders and that of requirements. To tackle the above challenge, Lim et al. [94,95] built a social network based on the recommendation of stakeholders. In this network, the stakeholders can be users, developers, legislators, and decision-makers. With this network, stakeholders and their requirements can be recommended and prioritized.

To determine developers' contributions to projects, Meneely et al. [17] presented two issue tracking ticket annotations to identify the originator and the approver in an issue. In their work, the collaboration information is combined with the version control system logs. Similarly, for identifying core developers of each release, Zhang et al. [32] extracted multi-types of DSNs from the ArgoUML mail archives based on topics. After an empirical study, they revealed that the metric of degree achieves the best performance of all the metrics.

To build successful software systems, more and more software projects require collaboration and communication of developers across the world during the entire software development lifecycle. Some studies related to developers' recommendation have been conducted in recent years. Ohira et al. [79] proposed a

tool, namely Graphmania, to visualize the relationship among developers and projects using collaborative filtering and social network analysis. Begel et al. [56] presented a recommending framework, namely Codebook, to mine flexibly transitive relationships between various artifacts of software repositories to support inter-team coordination needs. The DSN in Codebook can satisfy the need to recommend the counterparts of developers. Surian et al. [58] built a recommending system based on the projects and project properties to find reasonable developer candidates for a given developer. The Developer-Project-Property (DPP) graph is extracted as the network of developers' collaboration in their work.

6 Discussion

Based on the literature of DSNs, we investigate some key issues related to DSNs as follows.

Why DSNs work in resolving tasks in software engineering? As to our knowledge, there are two key reasons why DSNs can work in software engineering. First, some special information, which is not used in traditional software engineering, is captured in DSNs. Such new information includes churn information, dependency relationships, quantitative analysis of organization structures and ownership, tossing graphs. etc. With DSNs based on new information, one can find and mine some potential relationships revealing the essence of software development. Therefore, DSNs can be used to achieve better results for some specific tasks. For example, tossing graphs [75] can not only reduce the amount of bug tossing events but also improve the accuracy of automatic bug triage. Second, new advances in other social networks bring more new technologies and concepts for DSNs and eventually facilitate tasks in software engineering. For example, collaborative filtering is used in StakeRare [95] to avoid ignoring stakeholders.

When DSNs can be used in software engineering? DSNs have shown their effects in improving tasks in software engineering. To employ DSNs, some conditions must be satisfied. First, some information should be collected to construct DSNs. In the literature, a lot of information has been used, including email archives, source code, code change logs, bug report logs, etc. Second, all collected information should be related to tasks under resolving. For example, in the task of bug triage, the comments by developers on bug reports indicate common interests and skills of developers. Therefore, such comments could be employed to construct DSNs for recommending developers to a new bug report [57].

How DSNs can be used in software engineering? To employ DSNs for resolving tasks in software engineering, one could follow several steps. First, one should clarify the objective of the task and find related information for constructing DSNs. Then, one can investigate the hidden relationships between DSNs and the task under resolving. With DSNs, one can either resolve the task solely with DSNs or combine DSNs with existing approaches to facilitate the task. For example, in the phase of requirement analysis of the RALIC project [94], the objective is to identify and prioritize all stakeholders and the related information could be the initial set of stakeholders. After building the social network, Lim et al. [94] conducted a survey to request existing stakeholders to recommend new ones. Based on the survey, more complete social networks can be constructed and more stakeholders are involved.

7 Future work

The research area of DSNs cover a lot of disciplines, including linear and logic regression model, classification, collaborative filtering, and vector space model, etc. As to our knowledge, the research of DSNs consists of three parts, i.e., construction, analysis, and applications. Therefore, researchers can explore future directions in these parts successively.

New construction methods of DSNs. Since DSNs could be viewed as a special type of social networks, new techniques in constructing social networks could be extended to them. For example, the conformity phenomena in online social network have been considerably investigated in recent years. Tang et al. [96] defined three major types of conformities and quantify them by the Confluence model. Based on this idea, one can construct hybrid DSNs across multiple projects and use the Confluence model to

formalize the effects of social conformity and to predict user actions.

New analysis methods of DSNs. Some new analysis technologies arising in online social networks can be used in the context of DSNs. For example, it is a great challenge to determine the developers' collaboration in projects because many developers' links are hard to be captured [17]. In 2013, Kuo et al. [97] proposed an unsupervised framework to predict the links of unseen-type using aggregative statistics in heterogeneous social networks. The work of Kuo et al. [97] could be a choice to resolve the above challenge. Another example is to address the scalability problem of DSNs. Zhang et al. [32] identified the core developers of each release by extracting multi-types of DSNs from the ArgoUML mail archives. However, along with the growth of DSNs' sizes, the method proposed by Zhang et al. [32] becomes time consuming. A promising way to alleviate this problem is to employ Independent Path Algorithm (IPA), a scalable algorithm proposed by Kim et al. [98] to find the most influential nodes in a DSN.

New applications of DSNs. Nowadays, most applications of DSNs are conducted for software maintenance. In the future, one can investigate extending DSNs to the whole life cycles of software, including requirement analysis, design, implementation, and testing. For example, one may build up a DSN with collaborations among software architects, so as to recommend software architects for a new software project.

In addition, researchers may also investigate applications of DSNs in new areas of software engineering. For example, mobile software engineering [99–104], a new branch of software engineering, involves millions of mobile applications, millions of developers, and billions of users. In mobile software engineering, a DSN can be constructed based on similar mobile applications (developers who have contributed to similar applications can be linked). Then, a developer can achieve inspirations from similar open source projects as the new mobile application under development.

Acknowledgements

This work was supported by National Basic Research Program of China (973 Program) (Grant No. 2014CB3407-02), National Natural Science Foundation of China (Grant Nos. 61170067, 61373013, 61370144), and Scientific Research Foundation of Graduate School of Nanjing University (Grant No. 2013CL13).

References

1 Herbsleb J D, Mockus A. An empirical study of speed and communication in globally distributed software development. IEEE Trans Softw Eng, 2003, 29: 481–494
2 Cataldo M, Herbsleb J D. Communication networks in geographically distributed software development. In: Proceedings of 2008 ACM Conference on Computer Supported Cooperative Work, San Diego, 2008. 579–588
3 Manteli C, van Vliet H, van Den Hooff B. Adopting a social network perspective in global software development. In: Proceedings of 7th International Conference on Global Software Engineering, Porto Alegre, 2012. 124–133
4 Bird C. Sociotechnical coordination and collaboration in open source software. In: Proceedings of 27th IEEE International Conference on Software Maintenance, Williamsburg, 2011. 568–573
5 Brooks Jr, Frederick P. The Mythical Man-Month, Anniversary Edition: Essays on Software Engineering. New Jearsey: Pearson Education, 1995. 20–41
6 Madey G, Freeh V, Tynan R. The open source software development phenomenon: an analysis based on social network theory. In: Proceedings of Americas Conference on Information Systems, Dallas, 2002. 1806–1813
7 Xu J, Gao Y Q, Christley S, et al. A topological analysis of the open source software development community. In: Proceedings of 38th Hawaii International Conference on System Sciences, Hawaii, 2005. 198a–198a
8 Xu J, Christley S, Madey G. Application of social network analysis to the study of open source software. In: Jürgen B, Philipp J S, eds. The Economics of Open Source Software Development. Amsterdam: Elsevier Science, 2006. 205–224
9 Surian D, Lo D, Lim E P. Mining collaboration patterns from a large developer network. In: Proceedings of 17th Working Conference on Reverse Engineering, Beverly, 2010. 269–273
10 van Antwerp M, Madey G. The importance of social network structure in the open source software developer community. In: Proceedings of 43rd Hawaii International Conference on System Sciences, Hawaii, 2010. 1–10
11 Ferdian T, Bissyandé T F, Lo D, et al. Network structure of social coding in GitHub. In: Proceedings of 17th European Conference on Software Maintenance and Reengineering, Genova, 2013. 323–326

12 Lopez-Fernandez L, Robles G, Gonzalez-Barahona J M. Applying social network analysis to the information in CVS repositories. In: Proceedings of 2004 International Workshop on Mining Software Repositories, Edinburgh, 2004. 101–105
13 Yu L, Ramaswamy S. Mining CVS repositories to understand open-source project developer roles. In: Proceedings of 4th International Workshop on Mining Software Repositories, Minneapolis, 2007. 8–8
14 Meneely A, Williams L, Snipes W, et al. Predicting failures with developer networks and social network analysis. In: Proceedings of 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Atlanta, 2008. 13–23
15 Pohl M, Diehl S. What dynamic network metrics can tell us about developer roles. In: Proceedings of 2008 International Workshop on Cooperative and Human Aspects of Software Engineering, Leipzig, 2008. 81–84
16 Bettenburg N, Hassan A E. Studying the impact of social structures on software quality. In: Proceedings of 18th IEEE International Conference on Program Comprehension, Braga, 2010. 124–133
17 Meneely A, Corcoran M, Williams L. Improving developer activity metrics with issue tracking annotations. In: Proceedings of 2010 ICSE Workshop on Emerging Trends in Software Metrics, Cape Town, 2010. 75–80
18 Schwind M, Schenk A, Schneider M. A tool for the analysis of social networks in collaborative software development. In: Proceedings of 43rd Hawaii International Conference on System Sciences, Hawaii, 2010. 1–10
19 Jermakovics A, Sillitti A, Succi G. Mining and visualizing developer networks from version control systems. In: Proceedings of 4th International Workshop on Cooperative and Human Aspects of Software Engineering, Hawaii, 2011. 24–31
20 Meneely A, Williams L. Socio-technical developer networks: should we trust our measurements? In: Proceedings of 33rd International Conference on Software Engineering, Hawaii, 2011. 281–290
21 Jermakovics A, Sillitti A, Succi G. Exploring collaboration networks in open-source projects. In: Proceedings of 9th IFIP WG 2.13 International Conference, OSS 2013, Koper-Capodistria, 2013. 97–108
22 MacLean A C, Knutson C D. Apache commits: social network dataset. In: Proceedings of 10th IEEE Working Conference on Mining Software Repositories, San Francisco, 2013. 135–138
23 Simpson B. Changeset based developer communication to detect software failures. In: Proceedings of 35th International Conference on Software Engineering, San Francisco, 2013. 1468–1470
24 Crowston K, Howison J. The social structure of free and open source software development. First Monday, 2005, 10
25 Wagstrom P A, Herbsleb J, Carley K. A social network approach to free/open source software simulation. In: Proceedings of 1st International Conference on Open Source Systems, Genova, 2005. 16–23
26 Bird C, Gourley A, Devanbu P, et al. Mining email social networks. In: Proceedings of 2006 International Workshop on Mining Software Repositories, Shanghai, 2006. 137–143
27 Hossain L, Wu A, Chung K K. Actor centrality correlates to project based coordination. In: Proceedings of 2006 ACM Conference on Computer Supported Cooperative Work, Banff, 2006. 363–372
28 Bird C, Pattison D, D'Souza R. Latent social structure in open source projects. In: Proceedings of 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Atlanta, 2008. 24–35
29 Nia R, Bird C, Devanbu P, et al. Validity of network analyses in open source projects. In: Proceedings of 7th International Working Conference on Mining Software Repositories, Cape Town, 2010. 201–209
30 Canfora G, Cerulo L, Cimitile M, et al. Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD. In: Proceedings of 8th International Working Conference on Mining Software Repositories, Waikiki, 2011. 143–152
31 Zhang W, Yang Y, Wang Q. Network analysis of OSS evolution: an empirical study on ArgoUML project. In: Proceedings of 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution, Szeged, 2011. 71–80
32 Zhang W, Yang Y, Wang Q. An empirical study on identifying core developers using network analysis. In: Proceedings of 2nd International Workshop on Evidential Assessment of Software Technologies, Lund, 2012. 43–48
33 Sureka A, Goyal A, Rastogi A. Using social network analysis for mining collaboration data in a defect tracking system for risk and vulnerability analysis. In: Proceedings of 4th India Software Engineering Conference, Thiruvananthapuram, 2011. 195–204
34 Howison J, Inoue K, Crowston K. Social dynamics of free and open source team communications. In: Proceedings of IFIP Working Group 2.13 Foundation on Open Source Software, Como, 2006. 319–330
35 Datta S, Kaulgud V, Sharma V S, et al. A social network based study of software team dynamics. In: Proceedings of 3rd Annual India Software Engineering Conference, Mysore, 2010. 33–42
36 Hong Q N, Kim S, Cheung S C. Understanding a developer social network and its evolution. In: Proceedings of 27th IEEE International Conference on Software Maintenance, Williamsburg, 2011. 323–332
37 Zhou M, Mockus A. Does the initial environment impact the future of developers? In: Proceedings of 33rd International Conference on Software Engineering, Hawaii, 2011. 271–280
38 Kumar A, Gupta A. Evolution of developer social network and its impact on bug fixing process. In: Proceedings of 6th India Software Engineering Conference, New Delhi, 2013. 63–72
39 Xuan J F, Jiang H, Ren Z L, et al. Developer prioritization in bug repositories. In: Proceedings of 34th International

Conference on Software Engineering, Zurich, 2012. 25–35
40 Biçer S, Bener A B, Çağlayan B. Defect prediction using social network analysis on issue repositories. In: Proceedings of 2011 International Conference on Software and Systems Process, Honolulu, 2011. 63–71
41 Bhattacharya P, Neamtiu I. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In: Proceedings of 26th IEEE International Conference on Software Maintenance, Timisoara, 2010. 1–10
42 Bhattacharya P, Iliofotou M, Neamtiu I, et al. Graph-based analysis and prediction for software evolution. In: Proceedings of 34th International Conference on Software Engineering, Zurich, 2012. 419–429
43 Zhang T, Lee B. An automated bug triage approach: a concept profile and social network based developer recommendation. Intelligent Computing Technology, 2012. 505–512
44 Zanetti M S, Sarigol E, Scholtes I, et al. A quantitative study of social organisation in open source software communities. In: Proceedings of 7th International Conference on Computer Science & Education, Melbourne, 2012. 116–122
45 Zanetti M S, Scholtes I, Tessone C J, et al. Categorizing bugs with social networks: a case study on four open source software communities. In: Proceedings of 35th International Conference on Software Engineering, San Francisco, 2013. 1032–1041
46 Wu W J, Zhang W, Yang Y, et al. Drex: developer recommendation with k-nearest-neighbor search and expertise ranking. In: Proceedings of 18th Asia Pacific Software Engineering Conference, Ho Chi Minh City, 2011. 389–396
47 Nagappan N, Murphy B, Basili V. The influence of organizational structure on software quality: an empirical case study. In: Proceedings of 30th International Conference on Software Engineering, Leipzig, 2008. 521–530
48 Damian D, Helms R, Kwan I, et al. The role of domain knowledge and cross-functional communication in socio-technical coordination. In: Proceedings of 35th International Conference on Software Engineering, San Francisco, 2013. 442–451
49 Lim S L, Bentley P J. Evolving relationships between social networks and stakeholder involvement in software projects. In: Proceedings of 13th Annual Conference on Genetic and Evolutionary Computation, Dublin, 2011. 1899–1906
50 Wolf T, Schröter A, Damian D, et al. Predicting build failures using social network analysis on developer communication. In: Proceedings of 31st International Conference on Software Engineering, Vancouver, 2009. 1–11
51 de Souza C R, Quirk S, Trainer E, et al. Supporting collaborative software development through the visualization of socio-technical dependencies. In: Proceedings of 2007 International ACM Conference on Supporting Group Work, Sanibel Island, 2007. 147–156
52 Ell J. Identifying failure inducing developer pairs within developer networks. In: Proceedings of 35th International Conference on Software Engineering, San Francisco, 2013. 1471–1473
53 Pinzger M, Nagappan N, Murphy B. Can developer-module networks predict failures? In: Proceedings of 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Atlanta, 2008. 2–12
54 Bird C, Nagappan N, Gall H, et al. Putting it all together: Using socio-technical networks to predict failures. In: Proceedings of 20th International Symposium on Software Reliability Engineering, Mysuru, 2009. 109–119
55 Sarma A, Maccherone L, Wagstrom P, et al. Tesseract: interactive visual exploration of socio-technical relationships in software development. In: Proceedings of 31st International Conference on Software Engineering, Vancouver, 2009. 23–33
56 Begel A, Khoo Y P, Zimmermann T. Codebook: discovering and exploiting relationships in software repositories. In: Proceedings of 32nd International Conference on Software Engineering, Cape Town, 2010. 125–134
57 Surian D, Tian Y, Lo D, et al. Predicting project outcome leveraging socio-technical network patterns. In: Proceedings of 17th European Conference on Software Maintenance and Reengineering, Genova, 2013. 47–56
58 Surian D, Liu N, Lo D, et al. Recommending people in developers' collaboration network. In: Proceedings of 18th Working Conference on Reverse Engineering, Limerick, 2011. 379–388
59 Ricca F, Marchetto A. Heroes in FLOSS projects: an explorative study. In: Proceedings of 17th Working Conference on Reverse Engineering, Beverly, 2010. 155–159
60 Page L, Brin S, Motwani R, et al. The PageRank Citation Ranking: Bringing Order to the Web. Stanford: Stanford InfoLab. 1999
61 Marczak S, Damian D, Stege U. Information brokers in requirement-dependency social networks. In: Proceedings of 16th IEEE International Requirements Engineering Conference, Barcelona, 2008. 53–62
62 Tamburri D A, Lago P, van Vliet H. Uncovering latent social communities in software development. IEEE Softw, 2013, 30: 29–36
63 Wang X F, Chen G R. Complex networks: small-world, scale-free and beyond. Circ Syst Mag, 2003, 3: 6–20
64 Watts D J, Strogatz S H. Collective dynamics of small-world networks. Nature, 1998, 393: 440–442
65 Barabási A L, Albert R. Emergence of scaling in random networks. Science, 1999, 286: 509–512
66 Sharma V S, Kaulgud V. Studying team evolution during software testing. In: Proceedings of 4th International Workshop on Cooperative and Human Aspects of Software Engineering, Hawaii, 2011. 72–75
67 Duc A N, Cruzes D S, Ayala C. Impact of stakeholder type and collaboration on issue resolution time in OSS Projects. In: Proceedings of 7th IFIP 2.13 International Conference, OSS 2011, Salvador, 2011. 1–16
68 Gloor P A, Laubacher R, Dynes S B. Visualization of communication patterns in collaborative innovation networks-

analysis of some W3C working groups. In: Proceedings of 12th International Conference on Information and Knowledge Management, New Orleans, 2003. 56–60
69 Gilbert E, Karahalios K. CodeSaw: a social visualization of distributed software development. In: Proceedings of 11th IFIP TC 13 International Conference, Rio de Janeiro, 2007. 303–316
70 Borici A, Blincoe K, Schroter A. ProxiScientia: toward real-time visualization of task and developer dependencies in collaborating software development teams. In: Proceedings of 4th International Workshop on Cooperative and Human Aspects of Software Engineering, Zurich, 2012. 5–11
71 Wolf T, Schröter A, Damian D, et al. Mining task-based social networks to explore collaboration in software teams. IEEE Softw, 2009, 26: 58–66
72 Kwan I, Schroter A, Damian D. Does socio-technical congruence have an effect on software build success? A study of coordination in a software project. IEEE Trans Softw Eng, 2011, 37: 307–324
73 Hu W, Wong K. Using citation influence to predict software defects. In: Proceedings of 10th Working Conference on Mining Software Repositories, San Francisco, 2013. 419–428
74 Bird C, Nagappan N, Murphy B, et al. Don't touch my code! Examining the effects of ownership on software quality. In: Proceedings of 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, Szeged, 2011. 4–14
75 Jeong G, Kim S, Zimmermann T. Improving bug triage with bug tossing graphs. In: Proceedings of 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, Amsterdam, 2009. 111–120
76 Meneely A, Williams O. Interactive churn metrics: socio-technical variants of code churn. ACM SIGSOFT Softw Eng Notes, 2012, 37: 1–6
77 Zimmermann T, Nagappan N. Predicting defects using network analysis on dependency graphs. In: Proceedings of 30th International Conference on Software engineering, Leipzig, 2008. 531–540
78 Nguyen T H, Adams B, Hassan A E. Studying the impact of dependency network measures on software quality. In: Proceedings of 26th IEEE International Conference on Software Maintenance, Timisoara, 2010. 1–10
79 Ohira M, Ohsugi N, Ohoka T, et al. Accelerating cross-project knowledge collaboration using collaborative filtering and social networks. ACM SIGSOFT Softw Eng Notes, 2005, 30: 1–5
80 Fenton N E, Neil M. A critique of software defect prediction models. IEEE Trans Softw Eng, 1999, 25: 675–689
81 Nagappan N, Ball T, Zeller A. Mining metrics to predict component failures. In: Proceedings of 28th International Conference on Software Engineering, Shanghai, 2006. 452–461
82 Schröter A, Zimmermann T, Zeller A. Predicting component failures at design time. In: Proceedings of 2006 ACM/IEEE International Symposium on Empirical Software Engineering, Rio de Janeiro, 2006. 18–27
83 Weyuker E J, Ostrand T J, Bell R M. Using developer information as a factor for fault prediction. In: Proceedings of 3rd International Workshop on Predictor Models in Software Engineering, Minneapolis, 2007. 8–8
84 Cubranic D, Murphy G C, Singer J, et al. Hipikat: a project memory for software development. IEEE Trans Softw Eng, 2005, 31: 446–465
85 Bird C, Nagappan N, Devanbu P, et al. Does distributed development affect software quality? An empirical case study of Windows Vista. Commun ACM, 2009, 52: 85–93
86 Mockus A. Organizational volatility and its effects on software defects. In: Proceedings of 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Santa Fe, 2010. 117–126
87 Bettenburg N, Hassan A E. Studying the impact of social interactions on software quality. Empir Softw Eng, 2013, 18: 375–431
88 Amrit C, van Hillegersberg J, Kumar K. Identifying coordination problems in software development: finding mismatches between software and project team structures. arXiv:1201.4142, 2012
89 Rahman F, Devanbu P. Ownership, experience and defects: a fine-grained study of authorship. In: Proceedings of 33rd International Conference on Software Engineering, Hawaii, 2011. 491–500
90 Anvik J, Hiew L, Murphy G C. Who should fix this bug? In: Proceedings of 28th International Conference on Software Engineering, Shanghai, 2006. 361–370
91 Kagdi H, Hammad M, Maletic J I. Who can help me with this source code change? In: Proceedings of 24th IEEE International Conference on Software Maintenance, Beijing, 2008. 157–166
92 Cubranic D. Automatic bug triage using text categorization. In: Proceedings of 16th International Conference on Software Engineering & Knowledge Engineering, Banff, 2004
93 Xia X, Lo D, Wang X Y, et al. Accurate developer recommendation for bug resolution. In: Proceedings of 20th Working Conference on Reverse Engineering, Koblenz, 2013. 72–81
94 Lim S L, Quercia D, Finkelstein A. StakeNet: using social networks to analyse the stakeholders of large-scale software projects. In: Proceedings of 32nd International Conference on Software Engineering, Cape Town, 2010. 295–304
95 Lim S L, Finkelstein A. StakeRare: using social networks and collaborative filtering for large-scale requirements elicitation. IEEE Trans Softw Eng, 2012, 38: 707–735
96 Tang J, Wu S, Sun J. Confluence: conformity influence in large social networks. In: Proceedings of 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, 2013. 347–355

97 Kuo T T, Yan R, Huang Y Y, et al. Unsupervised link prediction using aggregative statistics on heterogeneous social networks. In: Proceedings of 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, 2013. 775–783

98 Kim J, Kim S K, Yu H. Scalable and parallelizable processing of influence maximization for large-scale social networks? In: Proceedings of 29th IEEE International Conference on Data Engineering, Brisbane, 2013. 266–277

99 Wasserman A I. Software engineering issues for mobile application development. In: Proceedings of FSE/SDP Workshop on Future of Software Engineering Research, Santa Fe, 2010. 397–400

100 Galvis Carreño L V, Winbladh K. Analysis of user comments: an approach for software requirements evolution. In: Proceedings of 35th International Conference on Software Engineering, San Francisco, 2013. 582–591

101 Gomez L, Neamtiu I, Azim T, et al. RERAN: timing-and touch-sensitive record and replay for Android. In: Proceedings of 35th International Conference on Software Engineering, San Francisco, 2013. 72–81

102 Hao S, Li D, Halfond W G. Estimating mobile application energy consumption using program. In: Proceedings of 35th International Conference on Software Engineering, San Francisco, 2013. 92–101

103 MacHiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for Android apps. In: Proceedings of 9th Joint Meeting on Foundations of Software Engineering, Saint Petersburg, 2013. 224–234

104 Jiang H, Ren Z L, Nie L M. Software engineering issues in mobile big data applications. Commun CCF, 2014, 10: 24–28