

1. Introduction

Software is a fundamental part of the modern life. We depend on the operation of software whether we are watching a video, traveling on a plane, calling a friend or using electricity power. Human life [1] and huge monetary losses [2] may be caused by issues that are not addressed in unreliable software components. Software quality assurance is the collection of activities that ensure the operations of software are complaint with the requirements [3]. Similar to all economical activities, software quality assurance is constrained by the budget of software projects. Efficient allocation of software quality assurance resources is crucial for the development of more reliable software. Development of a large software is beyond the technical capabilities of a single person or a few people. For this reason, large software is built and tested through the collaboration of many developers. Efficient coordination of the developers during the software quality assurance activities is one of the fundamental problems of software engineering.

Delivering a software bug-free is practically impossible since due to budget and time constraints exhaustive testing is not practical . Software are released with an unknown amount of inherent issues. It is a common belief in software engineering that as the number of eyeballs increase bugs become easier to notice [4]. Therefore after the release of a software, one of the most important part of software quality assurance is the publicly available issue management systems. In the issue management process, developers own (or get assigned to) issues reported by testing teams or software users. The key problem of this process is assigning the right developer to the right issue and showing the defect-prone parts of the software to the developers. In organizations twenty five percent of the time is spent while waiting for decisions [5]. We may improve the efficiency of issue management process by building an automated issue recommendation system to reduce the idle waiting time.

1.1. Software Networks

Large software can be classified as complex systems. Artificial and natural complex systems arise from the nature of the relations between many relatively simple entities [6]. Graph is a useful abstraction in computer science used to model complex systems and it is widely used in a range of application areas [7, 8, 9, 10]. In recent years, there has been considerable interest in modeling different complex systems as networks and various researchers examined the network properties in order to identify similarities among networks from diverse domains [9]. Scale-free networks are a group of networks from diverse domains that surprisingly share similar properties [8]. Networks such as academical collaboration, epidemics, world wide web and internet routers show some similar properties such as power-law degree distribution and low network diameter[8]. In the software domain, researchers have found that many dependency networks of large software systems are scale-free networks [11].

In software engineering, research on software networks goes back to early 1980s [12]. In the early years, most of the research has been on building the dependency graph for various programming languages and execution environments. Over the years, interest of researchers moved to investigating the relation of the dependency network and software quality. For this aim various researchers extracted metrics from dependency graphs based on common global and local properties of the graphs.

More recently, the edits on same software modules have been used to construct a collaboration network among developers similar to scientific collaboration network. This information has been used in empirical context in research on defect prediction models [13] and exploratory studies [14]. One reason behind this surge of interest may be the emergence of global software development [15]. Software network has been mostly studied for exploratory research previously. In this research, we used the software network to model the relation of issues with developers using the co-edits on same defective software modules related to the same category of issues to form the collaboration links among developers.

1.2. Issue Recommendation System

Recommendation systems are a subclass of information filtering system that seek to predict the rating or preference that a user would give to an item [16]. The recommendation system we propose is made of two core components namely bug triager and a live defect predictor. Bug triage is defined as assigning a developer to the most relevant issue based on past data [17]. Previous research shows that assigning a bug report to an owner takes five minutes per report on average for large software [17]. For popular projects, this task may make up the entire workload of several people. Previously, bug triage problem has been addressed by estimating the text similarity among issues [17]. This approach has various shortcomings such as language and issue report content dependence [17]. In our recommender, we categorize the issues based on their content markings. For each category, we rank the developers based on their collaboration activity and recommend new issues based on this ranking.

Live defect prediction is the prediction of defect-prone modules related to an issue. A defect predictor can be used to propose defect-prone modules to the assigned owner of an issue. In most defect prediction models the output of the model is provided at a special snapshot such as the production release or beginning of testing phase [18]. In our model we extend the machine learning based model by using the historical defect proneness data of related software modules using the call-graph of the software to propose defect prone modules *live* (at any point during development).

In order to build such a recommender, the mapping between the issues and defects has also been investigated in order to understand their relation with each other. In addition a new defect predictor has been built in order to detect defective modules live at any point during software maintenance.

If we examine the successful applications of AI that are widely used in practice, we observe that most of them need minimum interference by their users [16]. Successful applications such as movie recommenders, machine translators, image recognizers fit easily to the existing work flow of the users. Since the most common element in the

work flow of developers regarding quality assurance is issue management system, we believe that a practical recommendation system should fit easily to the issue management process. In order to make our issue recommender easily pluggable to the issue management software, we use data that can be extracted with automated tools from the issue and source code repositories.

1.3. Contributions

In this research we built an issue recommendation system using the collaboration network and the call graph of the software in order to increase the efficiency of resource allocation during the software quality assurance activities. Contributions of this research can be summarized as follows:

- *Model the software as the combination of developer collaboration network and the call graph:* We modeled software as the combination of developer collaboration and software call-graph networks. Collaboration is defined as the activity on the similar software modules by two developers during a release. Call graph is formed by the caller-callee relations of software modules. We showed the static and temporal properties of the software collaboration network for a large scale software and their relation with software issues.
- *Build an issue recommendation system using developer collaboration network:* Developer collaboration network shows the experience and centrality of developers within the organization. We used the symptom and category information of the issues to categorize the past issues and built the collaboration network for each cluster. Our model ranked the developers based on their centrality within the collaboration network and recommended the new issue reports based on the ranking.
- *Address developer workload balance problem using the issue recommendation model:* We extended the issue recommendation model to address the issue workload balancing problem. Issue workload balancing is a critical problem in real projects. We initially showed that in two large software, issues are distributed non-uniformly among maintainers especially during the times when active issue count is highest.

One problem of the base recommendation model was the lack of workload balance among developers that may occur when the recommendations of the model were used for issue assignment. We used an approach based on heuristics in order to overcome the problem of issue workload imbalances. Model that used workload balancing heuristics performed worse than the model with no heuristics but distributed issues more uniformly than the original model. Imbalances of the actual issue assignments may be the possible reason for the reduction of performance.

- *Usage of Kronecker networks to estimate the future collaborations of new developers:* Cold start is a common problem in recommender systems and it happens when a new user or item (issue) is added to the system. In our recommender, we solved the item(issue) cold start problem with a content based approach. We categorized the new issue into an existing category and afterwards ran the recommender algorithm.

Recommending new issues to developers is a more serious problem for the issue recommender. It is part of the problem of introducing new people to existing software development teams. As the project progresses initiation of new people gets harder. There is a lot of knowledge to be transferred to the new people reducing the productivity of both old and new employees during initiation. Fred Brooks claims that the addition of new people may even cause later project delivery dates [19].

We used the Kronecker network to address the new people cold start problem in the issue recommendation system. We used a parametric modelling method proposed by Leskovec Et al. [20, 21, 22] named Kronocker network to model the future state of the software collaboration network. We chose Kronecker graph generator since it is the only method that produce networks that conform all the properties of scale free networks. Kronecker networks are formed by fitting Kronecker parameters to the actual network and recursively replicating stochastic Kronecker kernel adjacency matrix by a matrix operation called Kronecker product.

The Kronecker network approach can be used to recommend issues only to a new group of developers since the node labels of the generated network can not be inferred. The state of the future collaboration network can be estimated using

the Kronecker parameters and issues can be recommended to the new group of developers.

- *Live defect prediction:* In traditional defect prediction models the defect prone-ness prediction for software modules is given for certain snapshots during the software project life-cycle such as the release revision or beginning of the testing phase to help managers allocate their testing resources. However, data that is comprised of defect prone modules is also valuable for developers at any time during maintenance. We updated the predictions of the machine learning based prediction model based on the live history of the software modules in the software call-graph to show the defect prone software modules any time during development.
- *Empirical analysis of the issue recommendation system and the live defect prediction model:* In our research, we collected the development data of one industrial large-scale software developed at five international sites as a dataset. The software is produced with the collaboration of more than two hundred developers. We extracted all the issue, collaboration activity and all the code change information over a time span of five years which includes two major releases of the product. We tested the issue recommendation model and its extensions on the extracted datasets and compared its performance with actual issue assignments. We showed that our model performance is better than the state of the art text similarity based issue recommendation systems. We showed the performance of Kronecker networks in addressing the people cold-start problem and the performance of the workload-balancing recommendation system extensions.

2. Problem Definition

Most of the management decisions in software engineering are based on the perceptions of the people about the state of the software and their estimations about its future states. Some of these decisions are; resource allocation, team building, budget estimation and release planning. There are many models and applications in both academia and industry that provide estimations about an aspect of the future state of a software [23]. While useful, most of these models do not recommend actions. In other words, these models mostly act like automated analysts but lack actual management support functionality. Two key reasons of this problem are modeling different aspects of software incompletely and providing output with little information content. Therefore, a problem in software engineering is modeling people, process and product (3P) with a more holistic approach.

In our work, we intend to provide a recommendation system whose output can be directly used to take actions. For this goal, we employ a model of the software that includes information about all aspects of 3P. The problem addressed by this dissertation can be divided into two dimensions: 1) Understanding the structure of software networks, 2) Building a recommender system that helps efficient issue management.

2.1. Understanding the Structure of Software Networks

In this dissertation work we model software projects as two distinct interlinked networks of developers and information elements (software modules) respectively. These two networks, namely the information network and collaboration network, can be seen in Figure 2.1. The *collaboration network* models the cooperation between developers, whereas the *information network* models the dependency relations among the software modules. These two networks together, provide us an abstraction of the software in the form of a *software network*.

The *information network* is a directed network which is based on the relations

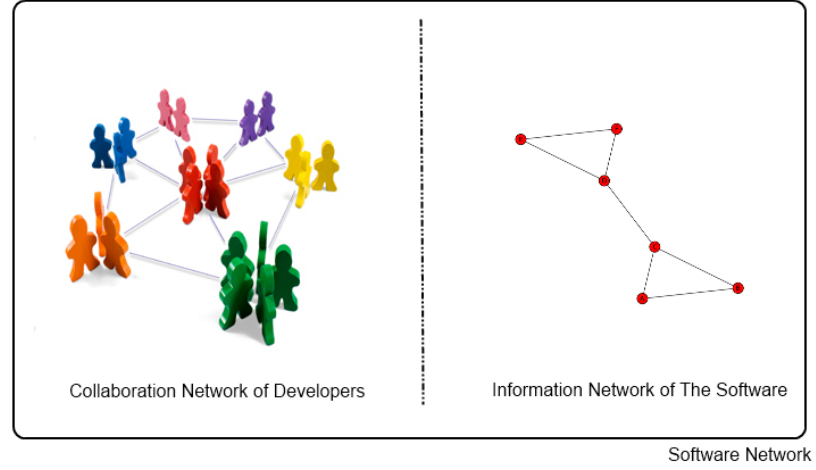


Figure 2.1: *Software network* is represented by the combination of its two underlying networks. a) *Collaboration network* models the interaction among developers, b) *Information network* models the relations between software components.

among software modules. An arc forms from software module a to software module b if a can call b at one point in its execution. The *collaboration network* is an un-directed network in which nodes are developers. An edge forms between two developers if they have contributed on a piece of software at one point.

The nodes in the collaboration network works on a subset of the information network of software modules at a given time . This information is represented in our model by links between software modules and contributor programmers. In the model, software quality is measured by number of defects found in a software module.

Once these two models have been built we aim to investigate the relation of information and collaboration networks in software with software quality. In addition, by modeling how the *software network* model evolves we aim to predict its future states. The *software network* model covers all aspects of 3P. The state of the *information network* provides information about the the product, the state of the *collaboration*

network provides information about people and the evolution of these networks provides information about the processes.

Previously these two networks were mined separately to the best of our knowledge. The evolution of these network has not been modelled either. One possible theoretical contribution of the dissertation will be modelling the *software network* and its evolution with a novel methodology.

The possible practical contributions to the field of software engineering will be as follows: a) We provide a novel useful model that covers 3P and that gives analysis results and predictions rich in information content, b) We model the future state of the software network to highlight future quality and collaboration problems.

2.2. Recommendation System For Issue Management

Defect prediction models provide a list of defect prone modules. However, they recommend no actions about how or by whom to handle them. While in theory providing successful results, defect prediction models face challenges when actively used in practice. One probable reason of these challenges is the difficulty of fitting defect prediction to the existing development environment in an organization.

Issue management is the central element of the software maintenance operations. A practical recommender system should be easily pluggable to the issue management system. Therefore, we propose a recommendation system that triages issues and recommends files that contain defects related to issues. In Figure 2.2 two main parts of the issue recommendation model can be observed. First part of such a system is the bug triage component. The problem of bug triage has been tackled by various researchers as a text mining problem previously [17] [24]. However, to the best of our knowledge no one used past development activity of developers and the issue timelines in triage. After assigning a bug to a developer, the first thing the developer will need would probably be the defective code relevant to the issue. In order to answer this problem we propose a live defect prediction model that predicts the defect prone modules in the

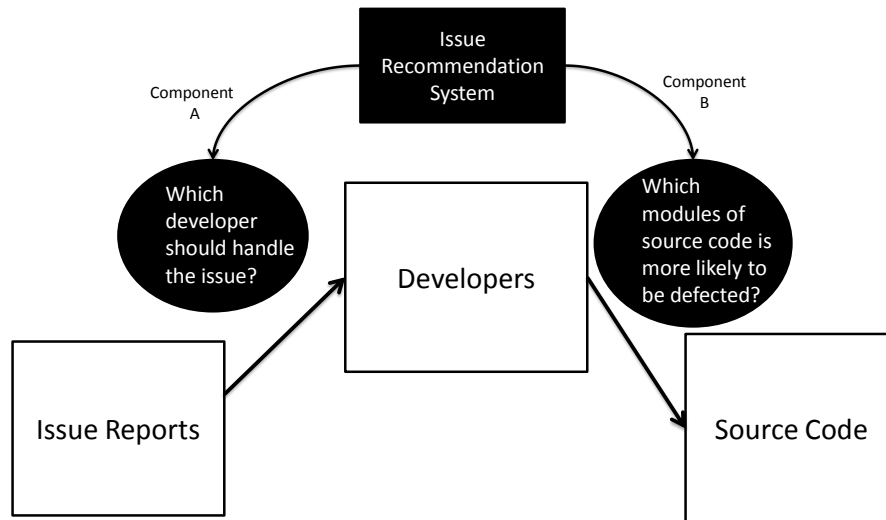


Figure 2.2: The recommendation system and its two components. Component A recommends an issue that is most relevant to a developer. Component B recommends the part of source code that possibly contains a defect related to that issue.

source code which are relevant to a developer. We believe that with its two components such a model would provide a major boost to the productivity of developers without requiring any input or change to their processes.

In order to develop such a recommendation system one needs to understand the relations between issues and defects and how the issue reports get generated. Therefore our research questions regarding the recommendation model are as follows:

- (i) How can we assign issues to developers?
- (ii) How can developers more efficiently organize their time in solving their issues?

REFERENCES

1. W. Everett and S. Honiden, “Reliability and safety of real-time systems,” *Software, IEEE*, vol. 12, no. 3, pp. 13–16, May.
2. B. Nuseibeh, “Ariane 5: Who dunnit?” *Software, IEEE*, vol. 14, no. 3, pp. 15–16, May-June 1997.
3. A. Puntambekar, *Software Engineering And Quality Assurance*. Technical Publications, 2010.
4. E. S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O’reilly and Associates, 1999.
5. U. Riss and A. Rickayzen, “Challenges for business process and task management,” ... *Knowledge Management*, vol. 0, no. 2, pp. 77–100, 2005. [Online]. Available: [http://www.jucs.org/jukm_0_2/riss/jukm_0_2_77_100_riss.html](http://www.jucs.org/jukm/_0_2/riss/jukm_0_2_77_100_riss.html)
6. J. H. Miller, *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*. Princeton University Press, 2007.
7. S. Brin and L. Page, “The anatomy of a large-scale hypertextual Web search engine,” *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107–117, Apr. 1998.
8. D. Easley and J. M. Kleinberg, *Networks , Crowds , and Markets : Reasoning about a Highly Connected World*, draft ed. Cambridge University Press, 2010.
9. J. M. Kleinberg, “Authoritative sources in a hyperlinked environment,” *Journal of the ACM*, vol. 46, no. 5, pp. 604–632, Sept. 1999. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=324133.324140>
10. A.-L. Barabási and E. Bonabeau, “Scale-free networks,” pp. 50–59, 2003. [Online].

Available: <http://elibrary.ru/item.asp?id=7821500>

11. L. Wen, R. G. Dromey, and D. Kirk, “Software engineering and scale-free networks.” *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 39, no. 4, pp. 845–54, Aug. 2009. [Online]. Available: <http://www.ncbi.nlm.nih.gov/pubmed/19380275>
12. F. Eichinger, K. Böhm, and M. Huber, “Mining edge-weighted call graphs to localise software bugs,” *Machine Learning and Knowledge Discovery in Databases*, pp. 333–348, 2008. [Online]. Available: <http://www.springerlink.com/index/26808jx13817xr73.pdf>
13. A. Meneely, L. Williams, W. Snipes, and J. Osborne, “Predicting failures with developer networks and social network analysis,” in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 13–23. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1453106>
14. G. Madey, V. Freeh, and R. Tynan, “The open source software development phenomenon: An analysis based on social network theory,” *Americas Conference on Information*, pp. 1806–1813, 2002. [Online]. Available: <http://ais.bepress.com/cgi/viewcontent.cgi?article=1606\&context=amcis2002>
15. J. Herbsleb and A. Mockus, “An empirical study of speed and communication in globally distributed software development,” *IEEE Transactions on Software Engineering*, vol. 29, no. 6, pp. 481–494, June 2003. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1205177>
16. D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender Systems: An Introduction*, 1st ed. Cambridge University Press, 2010.
17. J. Anvik, “Determining implementation expertise from bug re-

- ports,” *MSR '07 Proceedings of the Fourth International Workshop on Mining Software Repositories*, 2007. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1134285.1134457><http://dl.acm.org/citation.cfm?id=1808933><http://www.cs.ucdavis.edu/~barr/publications/fixation.pdf>http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4228639
18. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, “A Systematic Literature Review on Fault Prediction Performance in Software Engineering,” *IEEE Transactions on Software Engineering*, pp. 1–31, 2010.
 19. F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)*. Addison-Wesley Professional, 1995.
 20. J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, “Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication,” in *Knowledge Discovery in Databases: PKDD 2005*. Springer, 2005, pp. 133–145. [Online]. Available: <http://www.springerlink.com/index/ph80u3764t433020.pdf>
 21. J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graph evolution,” *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, pp. 2–41, Mar. 2007. [Online]. Available: <http://portal.acm.org/citation.cfm?doid=1217299.1217301>
 22. J. Leskovec, “Kronecker Graphs : An Approach to Modeling Networks,” *Journal of Machine Learning Research*, vol. 11, pp. 985–1042, 2010.
 23. T. Menzies, J. Greenwald, and A. Frank, “Data mining static code attributes to learn defect predictors,” *IEEE Transactions on Software Engineering*, vol. 33(1), pp. 2–13, 2007.
 24. A. Tamrawi, T. Nguyen, and J. Al-Kofahi, “Fuzzy set-based automatic bug triaging: NIER track,” *Proceedings of the 33rd International Conference on Software Engineering*, pp. 884–887, 2011. [Online]. Available: <http://>

[//ieeexplore.ieee.org/xpls/abs/_all.jsp?arnumber=6032543](http://ieeexplore.ieee.org/xpls/abs/_all.jsp?arnumber=6032543)