Effective Bug Triage – A Framework

V.Akila\textsuperscript{a}, Dr.G.Zayaraza, Dr.V.Govindasamy\textsuperscript{b}

\textsuperscript{a}Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry, India
\textsuperscript{b}Department of Information Technology, Pondicherry Engineering College, Pondicherry, India

Abstract

Bug resolution in Open Source Software is an important aspect of Open Source Software management. Bug resolution involves reporting a bug, triaging a bug and solving a bug. The task of solving a bug is analogous to problem solving in a Resolution Network. The duty of the triager is to identify the member with correct expertise to solve the bug as well as to identify any other member to whom the bug has to be tossed when the member is unable to solve the bug. The member though, may not have solved the bug, but may have made a significant contribution towards solving it in terms of fixing its severity, making comments, fixing its component etc. The work done by that member needs to be preserved. In addition, automated support is needed to find the optimal set of developers who can collaborate on a given bug. The order in which the bug can be tossed among the set of developers needs to be established. The current techniques in Open Source Software Bug Triaging involves modeling the reassignment of bugs as a goal-oriented path model. A new framework with the additional capabilities is proposed. The proposed framework models the reassignment of bugs as Enriched Adaptive Bug Triaging System (EABTS) based on actual path model.

Keywords:

1. Introduction

Open Source Software development is an intense distributed software development process. While Open Source Software development has thrown up rich repositories pertaining to the software development process like versioning repository, bug repository and email repository, it also has unwrapped some new challenges. Issue management is an important aspect of Open Source Software Maintenance. Software Bug Triaging is a vital step in Issue Management. Bug Triaging involves categorizing the bug, checking for validity, assigning severity level and most importantly assigning the bug to the appropriate developer. The process of assigning each software bug to the developer is fault prone and time consuming, if done manually. The contributors are at geographically separate
locations and different time zones. The interaction is mainly asynchronous. The set of active contributors varies from time to time. The expertise of contributors also varies with time. In this scenario, Issue Management is all the more intricate in Open Source Software development where it is difficult to keep track of the active developers. On an average, it takes 40 days to assign a bug to the correct developer. It takes another 100 days to reassign to a new developer if the first developer is unable to fix the bug [1]. It is essential to have automated support to recommend a developer for a reported bug. This reassignment of bugs from one developer to another is captured as a Bug Toss Graph based on Markov model [1].

The Bug Toss Graphs in Open Source Software are modelled as a goal-oriented graph [1]. In a goal-oriented graph, the developers toss relationship with a frequent resolver is captured. This graph essentially ignores information about the in between tosses between the developer and the resolver. While the overall intent of bug triaging is to reduce the number of tosses, there are some tosses which are useful for the resolving of the bug. The tosses that occur in the beginning of a bug’s life are beneficial in nature [2]. They are needed to fix the various fields in the bug report like the component, severity etc. The tosses that occur in the later part of the bug’s life are wasteful in nature. Reassignments are not always detrimental to bug-fix likelihood; several might be needed to find the optimal bug fixer [3]. Reassignments are needed to find the true cause of the bug and to find the optimal fixer [4].

Fig. 1. Frequency of Bug Field Modification in Kernel Project

Fig. 2. Frequency of Bug Field Modification in Mozilla Project
A bug is reported by a reporter. The bug is assigned to developer for resolution by the Triager. If the developer is not able to solve it, he may reassign or toss it to other developers. When tossing a bug, the developer may have fixed the component field or version field etc which was previously assigned wrongly. The analysis of the bug tosses for the period of January 2011 to December 2012 for Kernel project, Mozilla project and KDE project is shown in Fig. 1, Fig. 2 and Fig. 3 respectively. The bugs taken for consideration were resolved bugs with a vote count of ten. During the tosses, the fields modified are severity, component and version etc. The table shows the number of times the fields that were modified during bug tosses. While some modifications are wasteful, there are some modifications that are beneficial to the final fixing of the bug. This contribution by the developer is lost in the goal oriented path model. It may be argued that bug resolution is a collaborative activity among developers. It is crucial to retrieve the set of developers or experts sequentially with the right expertise. The goal here is to retrieve the experts such that the number of unproductive tosses is minimized.

To this end, the paper presents a framework for bug triaging based on an Enriched Relationship Graph (ERG).

**Actual Path Model**

The bug toss graph is modelled as a Markov model. Let $A=\{a_1, a_2, \ldots, a_n\}$ be a set of developers. The sequence $a_1 \rightarrow a_2 \rightarrow a_9$ is a tossing path, where $a_9$ is the resolver. Every single toss is considered. The adjacency matrix for set of tosses is formed. The bug toss graph modelled is a directed weighted graph. From the adjacency matrix, the transition probability of each toss is calculated. The transition probability matrix encodes the actual path model.

The transition probability $p(ij) = (\text{Weight of toss from } a_i \text{ to } a_j \div \text{Weight of tosses from } a_i)$.

The proposed graph structure is enriched because it not only captures the relationship among the developers as the number of tosses but also captures the propinquity that exists among the developers. The proposed technique is based on Ant routing which is inherently adaptive in nature. The outcome of the proposed work is a subgraph that consists of developers who are frequently involved in bug resolution.

2. Related Work

2.1. Bug Tossing in Open Source Software System

Pamela Battacharrya et al., [1] has proposed the idea of combining machine learning with bug tossing graph. An exploratory analysis of machine learning method was done and inferred that Naive Bayes produces the best results. The multi featured bug tossing graphs were modelled as goal oriented tossing graphs and a weight based searching algorithm was used to find the best toss relationship. The labels in the graph are the number of tosses, developers activity and the name of the product component. Liguo Chen et al., [5] combines bug tossing graphs with vector space model to identify developers. Here the feature used in the graph is the number of tosses. Gaeul Jeong et al., [6] was the first to employ bug tossing relationships in making developer recommendations. A goal oriented path model was made based on first order Markov model. Weighted Breadth first search algorithm was used to maximize the path reduction. Ashish Sureka [7] studies the component reassignment graphs. The graph contains nodes as
components and links as reassignments. The usefulness of component reassignment graph is investigated in the usefulness component reassignment prediction. Pamela Battacharrya et al.,[8] has visualized bug triaging as a two step activity , the first one being identification of the expert and the second one is finding the best tossing relationship based on goal oriented graph. Naïve Bayes classifier is used for machine learning. Incremental learning based on folding technique is used to achieve higher prediction accuracy.

2.2. Problem Resolution in Enterprise Networks

Qihong Shao et al., [9] address the problem of ticket routing by mining ticket resolution sequences. The ticket routing sequences are analyzed statistically and captured using Markov model. The order of the Markov model is chosen depending on the conditional entropy. The feature used in the graph is the number of tickets tossed. Peng Sun et al., [10] advocates for a hybrid model which retrieves resolution sequences that are similar to the new ticket. The weighted Markov model is created from this subset of resolution sequences which is used for generating routing recommendations. Gengxin Miao et al., [11] study the real life Collaborative networks to understand their characteristics. They have inferred that Collaborative Networks have truncated power law node degree distributions. A stochastic routing algorithm has been developed to simulate human dynamics in Collaboration networks. The algorithm is locally optimized. Yi Chen et al., [12] propose a computational framework to quantitatively assess expertise awareness. A novel Exclude algorithm has been designed that calculates transfer effectiveness. Gengxin Miao et al., [13] present a unified generative model that uses both content and routing sequence. A probabilistic algorithm has been developed that generates reliable ticket routing recommendations.

One can infer from the literature that ticket resolution in enterprise networks uses actual path model based on Markov model while in Open Source Software systems the tossing model is based on Goal oriented path model. Also the current techniques exploit only the number of tosses whereas an number of factors pertaining to relationship between the developers can be derived from the activity data. Also the Weighted Breadth First Search (WBFS) algorithm [14] that is used in the existing system is static in nature while the developer network for any OSS system tends to be dynamic. So the proposed framework models the underlying graph structure which models the proximity of one developer with another as an enriched structure and deploys the ant routing algorithm to find the path between any assignee and the final resolver.

3. Framework For Enriched Adaptive Bug Triaging System (EABTS)

The framework of the Enriched , Adaptive Bug Triage system(EABTS) is given in Figure 1. The EABTS is modelled as a learning system that consists of a Working Memory. After substantial learning is performed at the Working Memory, Consolidation takes place and knowledge is stored as a sub graph. In any learning system, the learner learns information from the data at disposal and stores it in a working memory. With time, the information that is revised is retained and remaining information is subject to trace decay. EATBS learns information regarding the optimal path from the first assignee to the final resolver. If the learned paths are frequently traced by the ant agents, then they are consolidated as knowledge in the sub graph otherwise they are decayed by gradual forgetting. The major modules in the framework are described as

- Data Preparation
- Working Memory

3.1. Data Preparation

The data preparation part consists of modelling the relationship among the developers as a ERG. The bugs are sourced from the bug repository such as www.bugzilla.com. The activity data of the bug report is extracted from the repository. From the activity data the ERG is modelled based on actual path model. The Enriched Relationship graph is multi featured and models the proximity of one developer with another. The labels in the graph are number of tosses, frequency, availability, recency, longevity and reciprocity[15].

Number of Tosses : Total number of tosses from one developer to another.
Longevity : The duration of relation between any two developer.
Recency : The freshness of the relation between any two developer.
Frequency : The number of tosses for the duration of relation between any two developer.
Availability : The existence of the developer
Reciprocity : The strength of the two way relation between any two developer.
Using these features the Enriched Relationship Graph serves as the underlying data structure for the bug triage process.

3.2. Working Memory

The working memory part consists of a learning model and a gradual forgetting model. The learning model uses ant agents to be deployed on the ERG. The ants lay pheromones on the paths to final resolver and routes are established. This activity goes on interactively so that the ants converge on a shortest path. This comprises the learning model. After the routes has been established, the Gradual Forgetting Model uses the trace decay by power law and trace decay by exponential law to implement the evaporation factor. The power law and exponential law decay needs to be explored because human forgetting is based on these laws. Finally, after a period of stabilization, the learned shortest paths are consolidated as knowledge.
4. Simulation Results

The proposed Enriched Adaptive Bug Triaging System (EABTS) has been partially implemented with number of tosses and longevity factors. The input to the system are the bug reports from 2009/01/09 and 2013/01/09 Eclipse project from www.bugzilla.org. Only the bugs with status “RESOLVED” and resolution as “FIXED” were considered. The experiments were run on a Pentium4 processor with 320 GB hard disc. Netbeans 7.2 was used as the frontend and Oracle as the backend. The performance of the EABTS system was compared with the baseline system which is based on Weighted Breadth First Search and which uses the number of tosses parameter. As it is observed from the figure 5. it is understood that EABTS-longevity gives superior performance with respect to the number paths with reduced number of hops.
5. Conclusion

A new enriched is proposed incorporating the ant routing algorithm. The proposed system preserves the work done by all the intermediate software developers. Further the system uses actual path model instead of goal oriented path model. Thus an efficient system which captures the knowledge of the shortest paths has been highlighted.

References


