A Decision Support Platform for Guiding a Bug Triager for Resolver Recommendation Using Textual and Non-Textual Features

Ashish Sureka, Himanshu kumar Singh, Manjunath Bagewadi, Abhishek Mitra, Rohit Karanth

Siemens Corporate Research and Technology, India

***Abstract*—Bug report assignment to a bug fixer or resolver is an important bug triaging activity. Bug assignment is a manual, time-consuming and tedious task. Research shows that manual bug assignment is often error-prone resulting in reassignments and bug tossing (an issue in both open-source and proprietary source projects). In this paper, we describe a decision support platform for guiding a bug triager for resolver recommendation. We design the decision support framework based on inputs from practitioners who are part of change control board or bug council of various product lines within a large engineering company. The novel aspects of our work in-context to existing work on automatic bug assignment is the application of non-textual features (in addition to the textual features consisting of terms in the bug reports) such as the software component, work-load of resolvers, role (such as triager, tester, product manager, test lead and developer) of a team-member within the life-cycle of a bug and the collaboration-network between the team members. The decision support system provides justification or reasoning, ranking and confidence score for every bug-report recommendation. We conduct experiments to evaluate the performance of our approach and share our deployment experience, insights and learning.**

Keywords—Bug Fixer Recommendation, Bug Triaging, Issue Tracking System, Machine Learning, Mining Software Repositories, Software Analytics, Software Maintenance

I. Problem Definition and Aim

Bug Resolver Recommendation, Bug Assignment or Triaging consists of determining the fixer or resolver of an issue reported to the Issue Tracking System (ITS). Bug Assignment is an important activity both in OSS (Open Source Software) or CSS/PSS (Closed or Proprietary Source Software) domain as the assignment accuracy has an impact on the mean time to repair and project team effort incurred. Identification of resolvers for open bug reports is normally done through a discussion between the committee members of a Change Control Board (CCB) representing various aspects of the software application such as project management, development, testing and quality control. Bug resolver assignment is non-trivial in a large and complex software setting (several bugs getting reported on a daily or weekly basis which increases the burden on the Triagers) consisting of distributed development. Bug assignment is a knowledge intensive task as it requires prior knowledge about the software system, expertise of the developer, team structure and composition and developer workload.

Research shows that manual assignment of bug reports to resolvers without any support from an expert system results in several incorrect assignments [1][2][3][4][5]. Incorrect assignment is undesirable and inefficient as it delays the bug resolution due to reassignments. Bug assignment is an important and a technically challenging task and hence attracted several researchers attention. While there has been recent advancements in solutions for automatic bug assignment, the problem is still not fully solved [1][2][3][4][5]. Furthermore, majority of the studies on automatic bug assignment are conducted on Open Source Software (OSS) data and there is a lack of empirical studies on Proprietary or Closed Source Software (PSS/CSS) data. In addition to lack of studies on Industrial or Commercial project data, application of non-textual features such as developer workload, experience and collaboration network for the task of automatic bug assignment is relatively unexplored. The work presented in this paper is motivated by the need to develop a decision support system for bug resolver recommendation based on the needs of Triagers. The specific research aim of the work presented in this paper are the following:

1) To investigate the application of textual (terms in bug reports) and non-textual features (components, developer workload, experience and collaboration network) for the task of automatic bug assignment using a Machine Learning based framework.
2) To conduct a survey of members from Change Control Board (CCB) to gain insights on factors influencing bug assignment.
3) To develop a web-based Decision Support System (DSS) guiding and assisting triagers for the task of automatic bug assignment as well gaining insights about the bug fixing efficiency, defect proneness and trends on time-to-repair through visual analytics and a dashboard.
4) To conduct empirical analysis on Closed Source or Proprietary Source Data (PSS/CSS) and demonstrate the effectiveness of the proposed approach.

II. Practitioner's Survey

Since our goal is to solve problems encountered by the practitioners, we conduct a survey of experienced practitioners to better understand their needs. We conduct a survey of 5 senior committee members belonging to Change Control Board (CCB) of our organizations software product lines. The average experience (in CCB) of the respondents was 7.5 years. In our organization, a CCB consists of members belonging to various roles: project manager, product manager, solution architect, QA (Quality Assurance) leader, developers

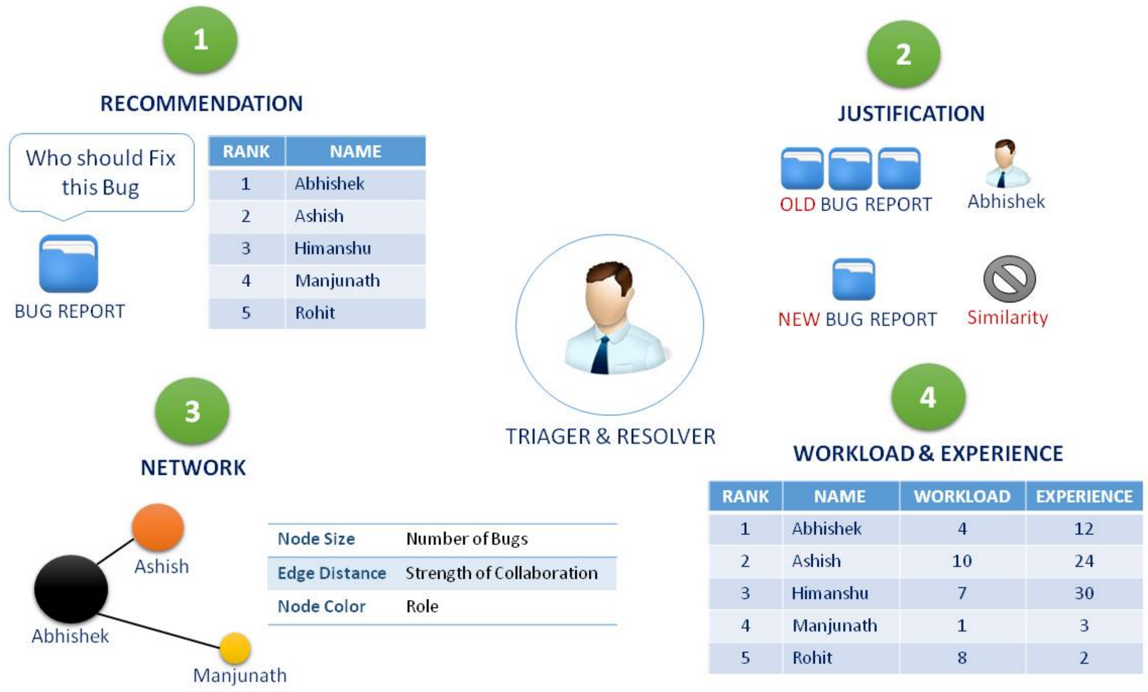


Fig. 2. A High-Level Overview of the Features: Top K Recommendation with Confidence or Score Value, Justification or Reasoning behind the Recommendation, Collaboration Network between the Developers, Workload & Prior Experience Values

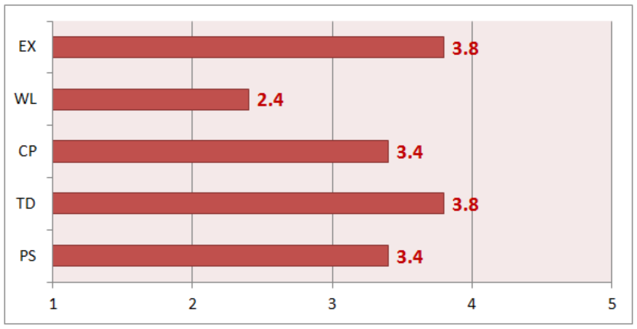


Fig. 1. Survey Results of Practitioners in Industry on Factors Influencing Bug Resolver Recommendation Decision [EX: Resolver Experience with the Project, WL: Resolver Workload, CP: Bug Report Component, TD: Bug Report Title and Description, PS: Bug Report Priority and Severity]

and testers. The survey respondents had been in various roles and active members of bug triaging process. Hence the survey responses are from representatives in-charge of various aspects such as development, quality control and management. The objective of our survey was to gain insights on factors influencing the change boards triaging decisions. Figure 1 shows the 5 questions in our questionnaire and the responses received. Each response is based on a 5 point scale (1 being low and 5 being high). Figure 1 reveals that there are clearly multiple factors and tradeoffs involved in making a triaging and bug assignment decision. We observe that bug report title and description and the available resolvers experience with the project are the two most important factors influencing the triaging decision (both having a score of 3.8 out of 5). The priority and severity of the bug as well as component assigned to the bug are also considered quite important with a score of 3.4. The current workload of the resolvers as a criteria influencing bug triaging decision received a score of 2.4 which is the lowest amongst all the 5 factors. The survey results support our objective of developing a bug resolver recommendation decision support system based on multiple factors (such as priority and severity of the bug report and current workload of the available resolvers) and not just based on matching the content of the bug report with the resolved bug reports of fixers.

III. User-Centered Design and Solution Architecture

We create a User-Centered Design (USD) keeping in mind the objectives and workflow of CCB. Our main motivation is to ensure a high degree of usability and hence we give extensive attention to the needs of our users. Figure 2 shows a high-level overview of the 4 features incorporated in our bug assignment decision support system. We display the Top K recommendation (k is a parameter which can be configured by the administrator) which is the primary goal of the recommender system. In addition to Top K recommendation, we present the justification and reasoning behind the proposed recommendation. We believe that displaying justification is important as the decision maker needs to understand the rule or logic behind the inferences made by the expert system. We display the textual similarity or term overlap and component similarity between the incoming bug report and the recommended bug report as justification to the end-user. We show developer collaboration network as one of the output of the recommendation system. The node size in the collaboration network represents the number of bugs resolved, edge distance or thickness represents the strength of collaboration (number of bugs co-resolved) and the node colour represents role. As shown in Figure 2, we display the developer workload and experience to the Triager

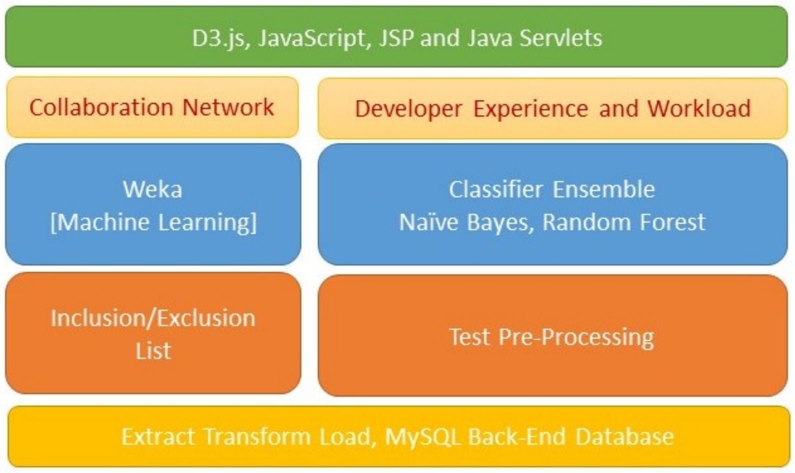

D3.js, JavaScript, JSP and Java Servlets
Collaboration Network
Developer Experience and Workload
Weka
[Machine Learning]
Classifier Ensemble
Naïve Bayes, Random Forest
Inclusion/Exclusion
List
Test Pre-Processing
Extract Transform Load, MySQL Back-End Database

Fig. 3. High-Level Architecture Diagram displaying Key Components (Front-End, Back-End and Middle Tier) - A Platform-Based Architecture

as complementary information assisting the user to make triaging decisions. Figure 2 illustrates all four factors influencing triaging decisions (Top K Recommendation, Justification and Reasoning, Collaboration Network and Developer Workload and Experience) which connects with the results of our survey and interaction with members of the CCB in our organization.

Figure 3 shows the high-level architecture illustrating key components of the decision support system. We adopt a platform-based approach so that our system can be customized across various projects using project based customization and fine-tuning. The architecture consists of a multi-step processing pipeline from data extraction (from the issue tracking system) as back-end layer to display as the front-end layer. As shown in Figure 3, we implement adaptors to extract data from Issue Tracking System (ITS) used by the project teams and save into a MySQL database. We create our own schema to save the data in our database and implement functionality to refresh the data based on a pre-defined interval or triggered by the user. Bug reports consists of free-form text fields such as title and description. We apply a series of text pre-processing steps on the bug report title and description before they are used for model building. We remove non content bearing terms (called as stop terms such as articles and propositions) and apply word stemming using the Porter Stemmer (term normalization). We create a domain specific Exclude List to remove terms which are non-discriminatory (for example, common domain terms like bug, defect, reproduce, actual, expected and behaviour). We create an Include List to avoid splitting of phrases into separate terms such as OpenGL Graphics Library, SQL Server and Multi Core. We first apply the Include List and extract important phrases and then apply the domain specific exclude list. The Include and Exclude List is customizable from the User Interface by the domain expert. The terms extracted from the title and description of the bug reports represents discriminatory features for the task of automatic bug assignment (based on the hypothesis that there is a correlation between the terms and the resolver).

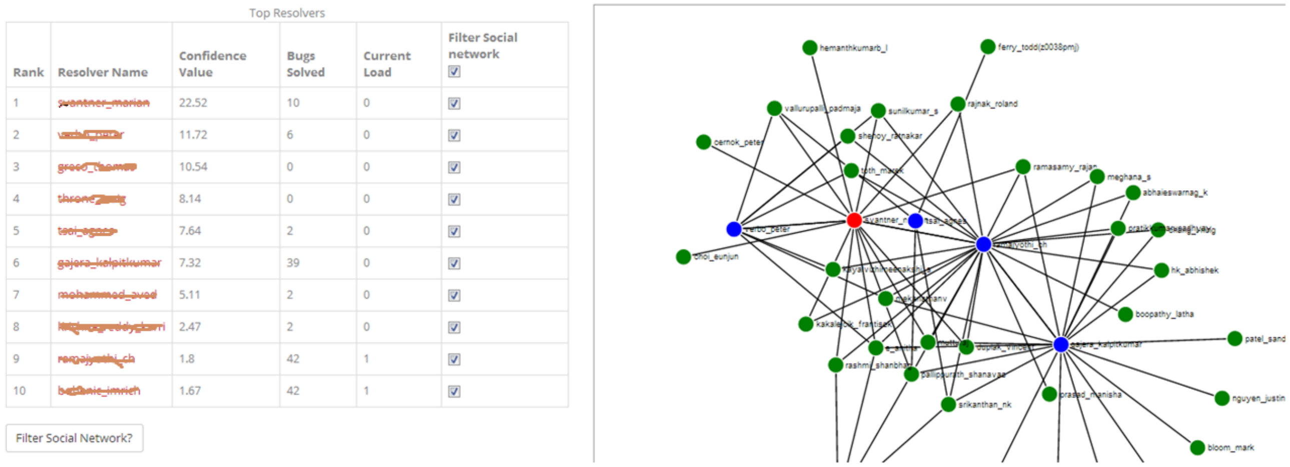
The next step in the processing pipeline is to train a predictive model based on the Machine Learning framework. We use a widely used Java based Machine Learning toolkit called as Weka[1] for model building and application. We embed Weka within our system and invoke its functionality using the Java API. We train a Random Forest and Nave Bayes classification model and use a voting mechanism to compute the classification score of the ensemble rather than individual scores to make the final predictions. We also extract the component of the bug report as a categorical feature as we observe a correlation between the component and the resolver. In terms of the implementation, we create an Attribute-Relation File Format (ARFF) that describes the list of training instances (terms and components and predictors and the resolver as the target class). As shown in the Figure 3, we extract the developer collaboration network, information on prior work experience with the project and workload from the Issue Tracking System (ITS). The ITS contains the number of bugs resolved by every developer from the beginning of the project. The ITS also contains information about the open bugs and the assignees for the respective open bug. We use close and open bug status information and the assignees field to compute the prior experience of a developer and the current work load with respect to bug resolution. Similarly, the collaboration network between developers is determined by extracting information from the bugs lifecycle. The frond-end layer implementation consists of D3.JS, JavaScript, Java Servlets and Java Server Pages (JSP).

We present our case study on a real-world project using the IBM Rational ClearQuest Issue Tracking System. ClearQuest keeps track of entire bug lifecycle (from reporting to resolution), state changes and comments posted by project team members. We consider three roles: triager, developer and tester. A comment can be posted and the state of a bug can be changed by triager, developer and tester. Figure 4 shows 9 possible states of a bug report in ClearQuest and the 91 possible transitions. A comment (in the ClearQuest Notes Log) consisting of state transition from Submitted to In-Work contains the triager and the developer role (from and to field). Similarly, In-Work to Solved state transition contains the developer and testers IDs. We parse ClearQuest Notes Log and annotate each project member ID with one of the three

[1]http://www.cs.waikato.ac.nz/ml/weka/

	Submitted	Qualified	In-Decision	Deferred	Terminated	In-Observation	In-Work	Solved	Validated
Submitted	TRG, TRG	TRG, TRG	TRG, TRG	TRG, TRG	DEV, TST	TRG, TRG	TRG, DEV	-	-
Qualified	-	DEV, DEV	TRG, TRG	TRG, TRG	TRG, TRG	TRG, TRG	TRG, DEV	-	-
In-Decision	-	-	DEV, DEV	DEV, TST	DEV, TST	DEV, TST	TRG, DEV	-	-
Deferred	-	-	TRG, TRG	TRG, TRG	TRG, TRG	TRG, TRG	TRG, TRG	-	-
Terminated	-	-	TST, TST	-	DEV, DEV	-	TST, DEV	-	-
In-Observation	-	-	TRG, TRG	TRG, TRG	DEV, TST	TST, TST	TST, DEV	-	-
In-Work	-	-	DEV, TST	DEV, TST	DEV, TST	DEV, TST	DEV, DEV	DEV, TST	-
Solved	-	-	-	-	-	-	TST, DEV	DEV, TST	TST, TST
Validated	-	-	-	-	-	-	TST, DEV	-	TST, TST

Fig. 4. List of 9 States in a Bug Lifecycle and 81 Possible Transitions. Infeasible Transitions are Represented by −. Each State Transitions is Used to Infer Roles within the Project Team. [TRG: Triager, DEV: Developer, TST, Tester]



Top Resolvers

Rank	Resolver Name	Confidence Value	Bugs Solved	Current Load	Filter Social network ☑
1	[redacted]	22.52	10	0	☑
2	[redacted]	11.72	6	0	☑
3	[redacted]	10.54	0	0	☑
4	[redacted]	8.14	0	0	☑
5	[redacted]	7.64	2	0	☑
6	[redacted]	7.32	39	0	☑
7	[redacted]	5.11	2	0	☑
8	[redacted]	2.47	2	0	☑
9	[redacted]	1.8	42	1	☑
10	[redacted]	1.67	42	1	☑

Fig. 5. A Snapshot of the Bug Resolver Recommendation Decision Support Tool displaying the Top 10 Recommendations, Confidence Value or Score, Workload and Past Experience with the Project

roles: developer, tester and triager. We then remove tester and triager and consider only the developers as bug resolvers for the purpose of predictive model building.

IV. Decision Support System User Interface

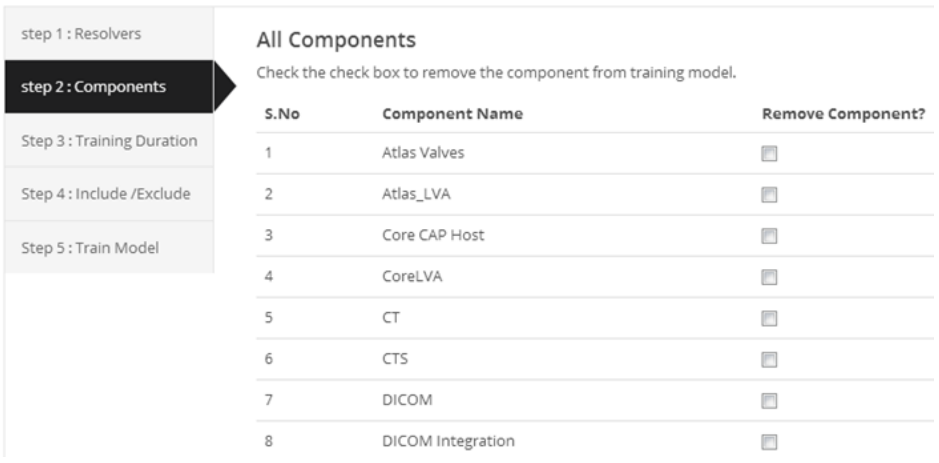
A. Recommendation and Settings

Figure 5 shows the snapshot of the decision support system displaying the Top K recommendation, score for each recommendation, prior work experience and the current work load of the proposed resolver. Figure 5 also shows the collaboration network of the developers. Nodes in the collaboration network can be filtered using the check-boxes provided in the screen. The confidence values shown in Figure 5 are probability estimates for each of the proposed resolver. The sum of the confidence values or probability estimates across all possible resolvers (and not just the Top K) sum up-to 1. We display the probability estimates and not just the rank to provide additional information on the strength of the correlation between the resolver and the incoming bug report. Figure 6 shows a snapshot of the settings page consisting of five tabs: resolvers, components, and training duration, include and exclude list and train model. We describe and provide a screenshots for one of the tabs due to limited space in the paper. We apply a platform-based approach and provide a configurable settings page so that the decision support system can be customized according to specific projects.

As shown in Figure 6, a user can add, rename and modify components component names. A software system evolves over a period of time and undergoes architectural changes. New components get added, components gets merged and renamed. We provide a facility to the user to make sure that the model built on the training data is in-synch with the software system architecture. Similar to component configuration, we provide a tab to customize resolver list. For example, if a developer has left the organization then its information can be deleted through the Resolver tab and ensure that his or her name is not shown in the Top K recommendation. The training instances and the amount of historical data on which to train the predictive model can also be configured. The predictive model should be representative of the current practise and hence we provide a facility for the user to re-train the model based on recent dataset.

B. Visual Analytics on Bug Resolution Process

In addition to the Top K recommendation, justification, developer collaboration network, developer prior work experience and current workload, we also present interactive visualizations on the bug resolution process. Francalanci et al. [6] present an analysis of the performance characteristics (such as continuity and efficiency) of the bug fixing process. They identify performance indicators (bug opening and closing trend) reflecting the characteristics and quality of bug fixing



step 1 : Resolvers

step 2 : Components

Step 3 : Training Duration

Step 4 : Include /Exclude

Step 5 : Train Model

All Components

Check the check box to remove the component from training model.

S.No	Component Name	Remove Component?
1	Atlas Valves	☐
2	Atlas_LVA	☐
3	Core CAP Host	☐
4	CoreLVA	☐
5	CT	☐
6	CTS	☐
7	DICOM	☐
8	DICOM Integration	☐

Fig. 6. A Snapshot of the Setting Page Consisting of 5 Tabs: Resolvers, Components, Training Duration, Include & Exclude List, Train Model. The Spanshot displays List of Components and the Remove Option

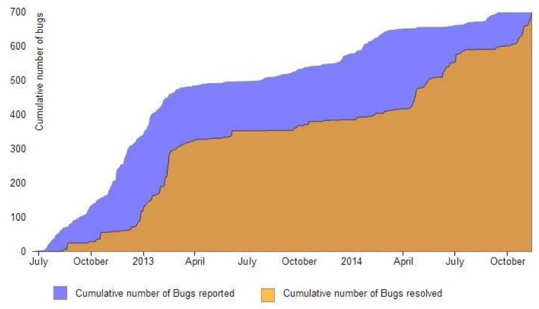


Fig. 7. Graph Depicting Bug Fix Quality as the Extent of Gap between the Bug Opening and Bug Closing Trend or Curve

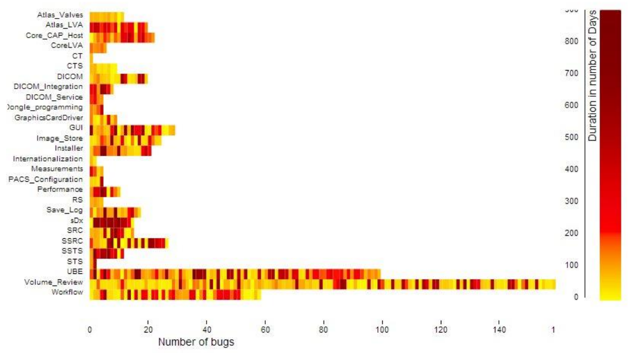


Fig. 8. A Combination of a Heat Map and a Horizontal Bar Chart displaying Three Dimensions in One Chart: Component, Number of Bugs and Duration to Resolve the Bug

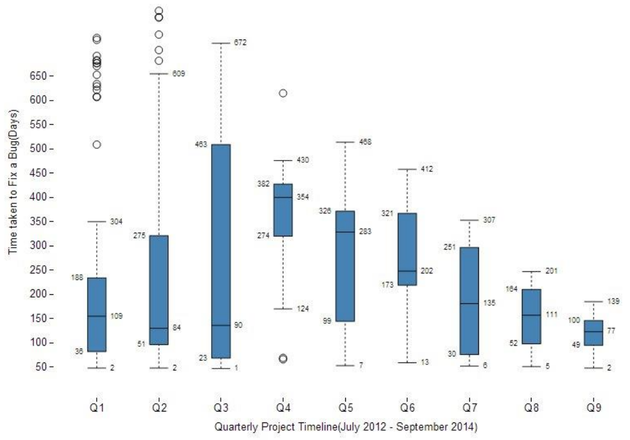


Fig. 9. A Spectrum of Boxplots Depicting Descriptive Statistics on Time Taken to Fix a Bug. Each Boxplot Corresponds to Dataset Belonging to One Quarter

process. We apply the concepts presented by Francalanci et al. [6] in our decision support system. They define bug opening trend as the cumulated number of opened and verified bugs over time. In their paper, closing trend is defined as the cumulated number of bugs that are resolved and closed over time. Figure 7 displays the opening and closing trend for the Issue Tracking System dataset used in our case-study. At any instant of time, the difference between the two curves (interval) can be computed to identify the number of bugs which are open at that instant of time. We notice that the debugging process is of high quality as there is no uncontrolled growth of unresolved bugs (the curve for the closing trend grows nearly as fast or has the same slope as the curve for the opening trend). Figure 8 shows a combination of Heat Map and a Horizontal Bar Chart providing insights on the defect proneness of a component (in-terms of the number of bugs reported) and the duration to resolve each reported bug. We observe that the bug fixing time for the Atlas Valves component is relatively on the lower side in comparison to the sDx component. UBE, Volume Review and Workflow are the three components on which maximum numbers of bugs have been reported. The information presented in Figure 8 is useful to the CCB as the bug resolver recommendation decision is also based on the buggy component and the defect proneness of the component. Figure 9 shows a spectrum of Boxplots across various years and quarters displaying descriptive statistics and five-number summary on time taken to fix a bug (bug resolution time). The spectrum of Boxplots provides insights to the CCB on the changes in the distribution of resolution time over several quarters or time periods.

Figure 10 shows a bubble chart displaying component diversity and trends on the average number of developers

TABLE I. RECALL@K , PRECISION@K AND F-MEASURE@K FOR XANADU PROJECT

	K=1	K=2	K=3	K=4	K=5	K=6	K=7	K=8	K=9	K=10
RECALL	0.160	0.337	0.474	0.547	0.599	0.647	0.688	0.721	0.750	0.774
PRECISION	0.324	0.353	0.324	0.287	0.254	0.232	0.214	0.199	0.186	0.174
F-MEASURE	0.214	0.345	0.385	0.376	0.357	0.342	0.327	0.313	0.298	0.284

TABLE II. RECALL@K , PRECISION@K AND F-MEASURE@K FOR S2000 PROJECT

	K=1	K=2	K=3	K=4	K=5	K=6	K=7	K=8	K=9	K=10
RECALL	0.242	0.644	0.794	0.819	0.848	0.864	0.875	0.890	0.900	0.905
PRECISION	0.437	0.599	0.493	0.408	0.342	0.292	0.255	0.228	0.206	0.188
F-MEASURE	0.312	0.620	0.609	0.545	0.488	0.436	0.395	0.363	0.335	0.311

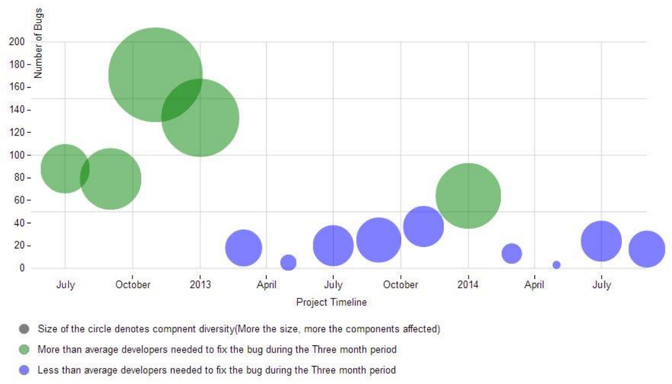


Fig. 10. A Bubble Chart displayed as a Scatter Chart in which each Data Point denotes Component Diversity (Bubble Size) and Number of Resolvers (Color) across Various Quarters

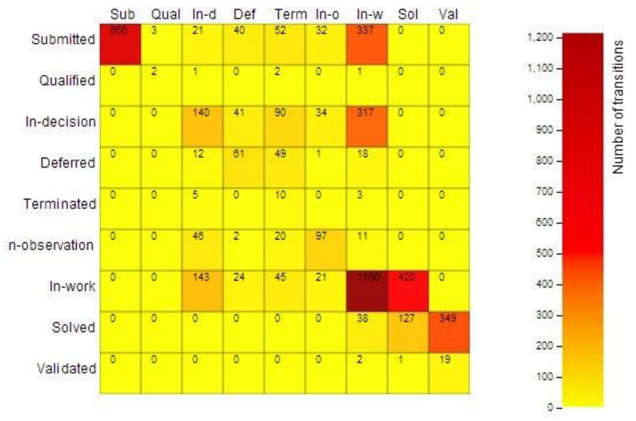


Fig. 11. A Heat Map showing the Number of Transitions (during the Bug Lifecycle of all Bug Reports in the Dataset) between the 8 Possible States

needed to a resolve a bug across project time-line. Figure 10 reveals that the component diversity was high in July and October Quarter of the year 2013 which means that the reported bugs were spread across various components. We infer that the component diversity decreases in April and July Quarter for the year 2014 which means that majority of the bugs were reported within a small number of components. We also present insight on average number of developers needed to resolve a bug. We first compute the average number of developers needed to resolve a bug over the entire 2 years (dataset period) and then color-code the bubble for each quarter depending on its value being above or below the average value. Figure 11 displays the number of state transitions between any of the 91 state transitions. Figure 11 is a Heat Map in which every cell is color coded depending on the number transitions representing the cell. The Heap Map is useful to the CCB in gaining insights on process anti-patterns and inefficiencies. For example, Reopened bugs increase the maintenance costs, degrade overall user-perceived quality of the software and lead to un-necessary rework by busy practitioners [7]. Figure 11 reveals several cases of bug re-opening (such Solved to In-work, Terminated to In-Decision).

V. EMPIRICAL ANALYSIS AND RESULTS

VI. CONCLUSIONS

Our survey results demonstrate that there are multiple factors influencing triaging decision. Terms in bug report title and description as well as resolver experience with the project are the two most important indicators for making bug assignment decision. Our interaction with practitioners in our organization reveals that justification or reasoning behind a recommendation, developer collaboration network, developer work experience and workload are also important and useful information in addition to the Top K recommendation. Descriptive statistics, trends and graphs on bug fixing efficiency, big opening and closing trends, mean time to repair and defect proneness of components are also important and complementary information for the Change Control Board while making triaging decisions. We demonstrate the effectiveness of our approach by conducting experiments on real-world CSS/PSS data from our organization and report encouraging accuracy results. We conclude that an ensemble of classifiers consisting of Decision Tree and Nave Bayes learners is an effective mechanism for the task of automatic bug assignment.

REFERENCES

[1] G. Bortis and A. v. d. Hoek, “Porchlight: A tag-based approach to bug triaging,” in *Proceedings of the 2013 International Conference on Software Engineering*, ICSE ’13, pp. 342–351, 2013.

[2] X. Xia, D. Lo, X. Wang, and B. Zhou, “Accurate developer recommendation for bug resolution,” in *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pp. 72–81, 2013.

[3] X. Xie, W. Zhang, Y. Yang, and Q. Wang, “Dretom: Developer recommendation based on topic models for bug resolution,” in *Proceedings of the 8th International Conference on Predictive Models in Software Engineering*, PROMISE ’12, pp. 19–28, 2012.

[4] W. Wu, W. Zhang, Y. Yang, and Q. Wang, “Drex: Developer recommendation with k-nearest-neighbor search and expertise ranking,” in *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*, pp. 389–396, 2011.

[5] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, “Fuzzy set and cache-based approach for bug triaging,” in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, pp. 365–375, 2011.

[6] C. Francalanci and F. Merlo, “Empirical analysis of the bug fixing process in open source projects,” in *Open Source Development, Communities and Quality*, pp. 187–196, 2008.

[7] E. Shihab, A. Ihara, Y. Kamei, W. Ibrahim, M. Ohira, B. Adams, A. Hassan, and K.-i. Matsumoto, “Studying re-opened bugs in open source software,” in *Empirical Software Engineering*, pp. 1005–1042, 2013.