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*—It is largely believed among researchers that the software engineering methods and techniques based on mining of software repositories (MSR) have the potential of providing sound and empirical basis for Software Engineering tasks. But it has been observed that the main hurdles to adoption of the techniques are organizational in nature or people centric, for example lack of access to data, organizational inertia, general lack of faith in results achieved without human intervention, and a tendency of experts to feel that their inability to arrive at optimal decisions is rooted in someone else's shortcomings, in this case person who files the bug. We share our experiences in developing a use case for applying such methods to the common software engineering task of Bug Triaging within an industrial setup. We accompany the well researched technique of applying textual information content in bug reports with additional measures in order to improve the acceptance and effectiveness of the system. Specifically we present: A) use of non-textual features for factoring in the decision making process that a human would follow; B) making available effectiveness metrics that present a basis for comparing the results of the automated systems against the existing practice of relying on human decision making; and C) presenting reasoning or the justification behind the results so that the human experts can validate and accept the results. We present these non-textual features and some of the metrics and discuss on how these can address the adoption concerns for this specific use case.**

***Index Terms*—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. Bug resolver assignment is non-trivial in a large and complex software setting, especially with globally distributed teams, wherein several bugs may get reported on a daily or weekly basis increasing the burden on the triagers. One of the primary ways, identification of resolvers for open bug reports is normally done is through a decision by Change Control Board (CCB), members of which represent various aspects of the software project such as project management, development, testing and quality control. The CCB usually works as a collective decision making body that reviews the incoming bugs and decides who to assign it to, or whether more information is required, or a bug is irrelevant, or behavior needs to be observed more. As can be imagined that the decisions made by the CCB are knowledge intensive and it requires prior knowledge about the software system, expertise of the developer, team structure and composition and developer workload. In some instances, in order to optimize the time of the entire CCB, a pre-CCB is conducted by individual members of the board on assigned subset of bugs and the individual recommendations are reviewed in complete CCB. The average time to triage a bug in such a process can be captured as following:

$$t = \frac{T_{pre-CCB} * M + T_{CCB}}{N}$$

Here, t denotes the average time it takes to triage a bug, given M committee members taking $T_{pre-CCB}$ time individually for assessing their subset of bugs and T_{CCB} time together to discuss and finalize recommendation for N bugs. From the above, its clear that any method that can assist in reducing any of M, T_{CCB} and $T_{pre-CCB}$ has potential to increase overall efficiency. 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. 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 OSS data and there is a lack of empirical studies on 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 aims of the work presented in this paper are:

1) To build a decision support system for guiding and assisting triagers for the task of automatic bug assignment, this involves application of textual (terms in bug reports) to build a classification model

2) Using of non-textual features (components, developer workload, experience, collaboration network, process map) for contextualizing the model.
3) Provide insights about the bug fixing efficiency, defect proneness and trends on time-to-repair through visual analytics and a dashboard.
4) Build the system in user centric manner by providing the justification and reasoning behind the recommended assignments.

Rest of the paper is structured as follows: in Section II we discuss and argue that user centric approach to build such recommendation systems incorporates the elements necessary to address the above goals. Next we discuss some of the contextualization measures for the model, specifically use of a practitioner survey results and process map. In Section IV, describes some of the metrics and measures that accompany the system are how can they be used. Section V presents early results from applying the system on two sets of data obtained from actual industrial projects, one active for 2 years whereas other for 9 years.

II. User centered Design and Solution Architecture

We create a User-Centered Design considering 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 1 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 color represents role. As shown in Figure 1, we display the developer workload and experience to the Triager as complementary information assisting the user to make triaging decisions. Figure 1 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 2 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 2, 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 consist 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 behavior). 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. Include and Exclude Lists are 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 used Weka which is a widely used Java based Machine Learning toolkit called 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 Naive 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 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 Servlet and Java Server Pages (JSP).

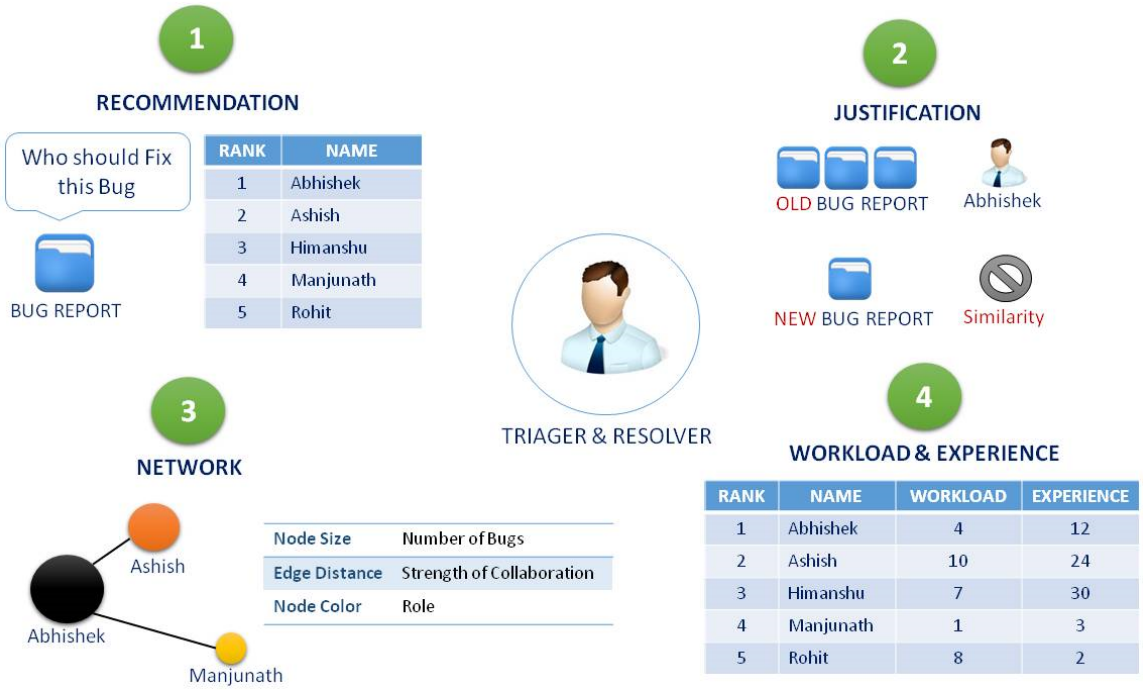

Fig. 1. A High-Level Overview of the Features: Top K Recommendation with Confidence or Score Value, Justification or Reasoning behind the Recommendation, Collaboration Network between the Developers, Workload & Prior Experience Values

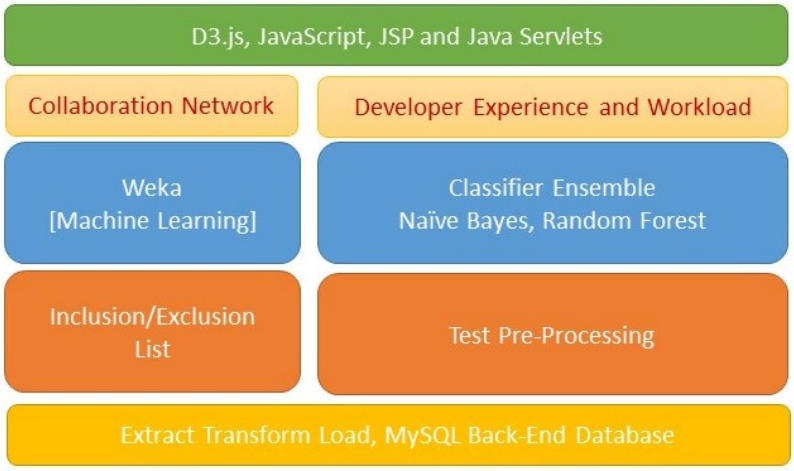

Fig. 2. High-Level Architecture Diagram displaying Key Components (Front-End, Back-End and Middle Tier) - A Platform-Based Architecture

III. Model Contextualization and Process Parameters

Since our goal is to solve problems encountered by the practitioners and model the system as closely as possible to the actual process and workflows of CCB, 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, quality assurance leader, developers, 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.

IV. Practitioner's Survey

Figure 3 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 3 reveals that

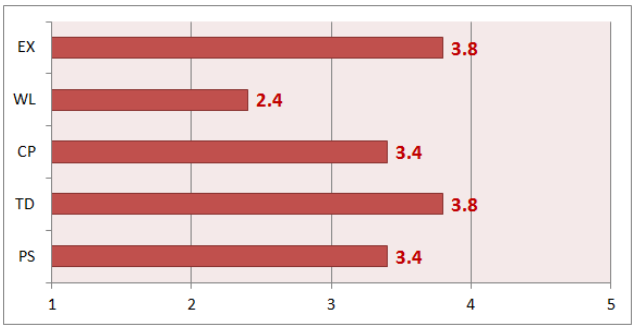

Fig. 3. Survey Results of Practitioners in Industry on Factors Influencing Bug Resolver Recommendation Decision [EX: Resolver Experience with the Project, WL: Resolver Workload, CP: Bug Report Component, TD: Bug Report Title and Description, PS: Bug Report Priority and Severity]

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 out of 5 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.

We present our case study on a real-world project using the IBM Rational ClearQuest as 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 81 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 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. This step of inferring developers is crucial since, triagers and testers frequently commit on the bug reports and their comments should not skew the results.

V. 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 upto 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 practice 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 [6], developer prior work experience and current workload, we also present interactive visualizations on the bug resolution process. Francalanci et al. [7] 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 process. We apply the concepts presented by Francalanci et al. [7] 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 [7][8].

Figure 7 displays the opening and closing trend for the Issue Tracking System dataset used in our case-study. At any instant

	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]

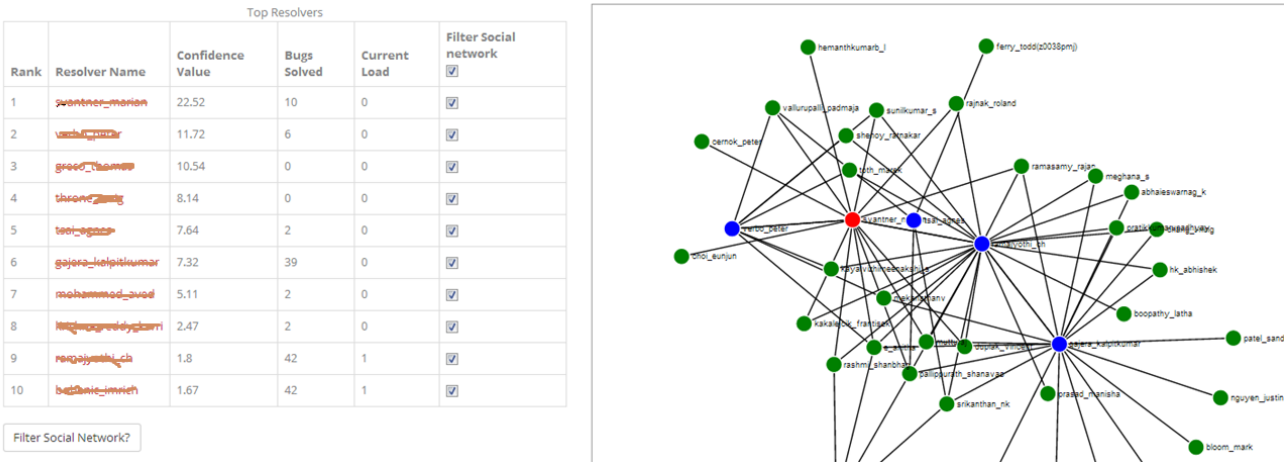

Fig. 5. A Snapshot of the Bug Resolver Recommendation Decision Support Tool displaying the Top 10 Recommendations, Confidence Value or Score, Workload and Past Experience with the Project

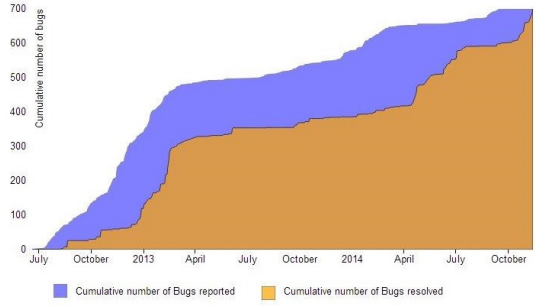

Fig. 7. Graph Depicting Bug Fix Quality as the Extent of Gap between the Bug Opening and Bug Closing Trend or Curve

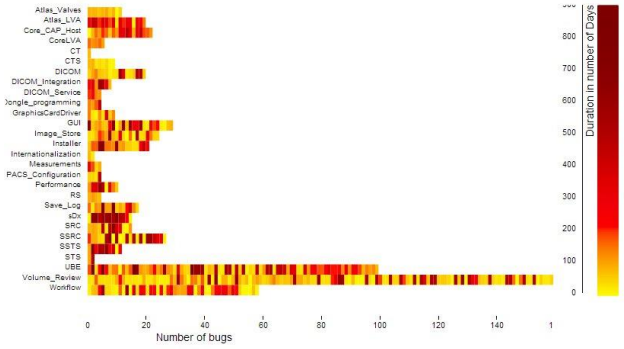

Fig. 8. A Combination of a Heat Map and a Horizontal Bar Chart displaying Three Dimensions in One Chart: Component, Number of Bugs and Duration to Resolve the Bug

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 7 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

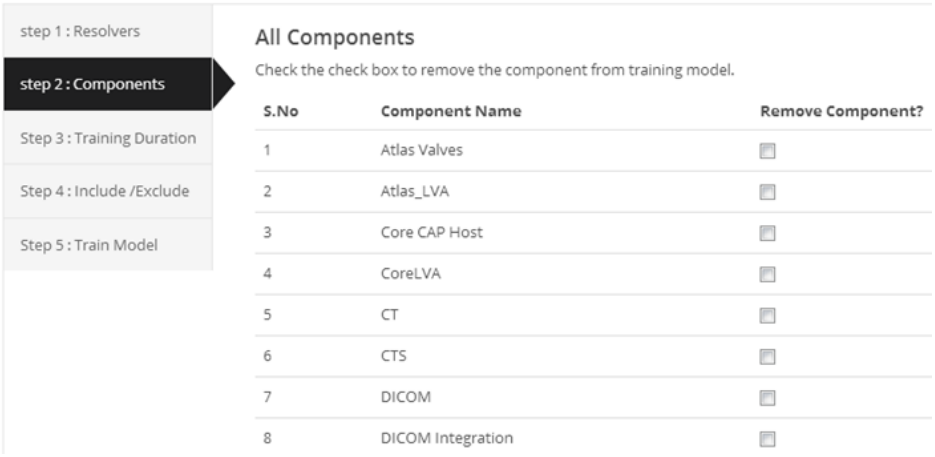

Fig. 6. A Snapshot of the Setting Page Consisting of 5 Tabs: Resolvers, Components, Training Duration, Include & Exclude List, Train Model. The Spanshot displays List of Components and the Remove Option

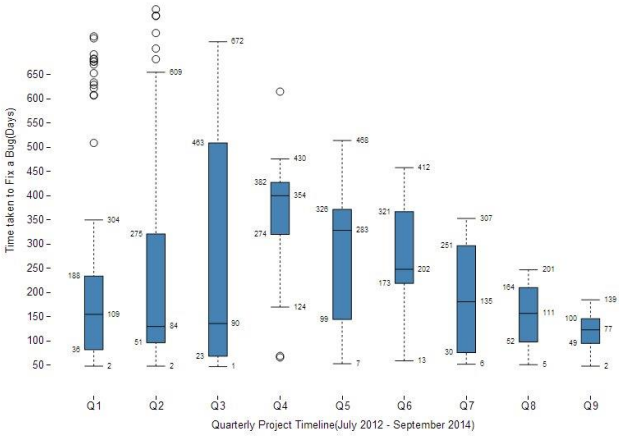

Fig. 9. A Spectrum of Boxplots Depicting Descriptive Statistics on Time Taken to Fix a Bug. Each Boxplot Corresponds to Dataset Belonging to One Quarter

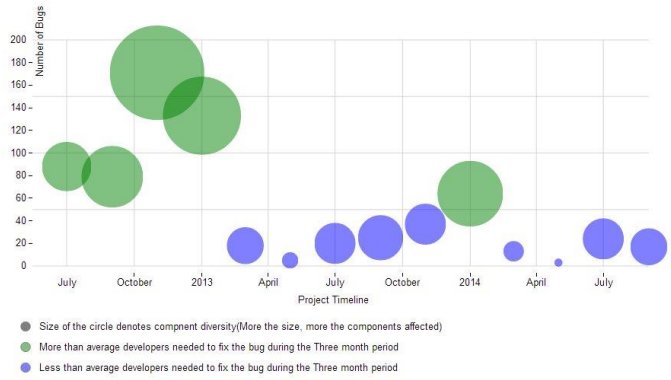

Fig. 10. A Bubble Chart displayed as a Scatter Chart in which each Data Point denotes Component Diversity (Bubble Size) and Number of Resolvers (Color) across Various Quarters

buggy component and the defect proneness of the component. Figure 9 shows a spectrum of Box plots 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 Box plots 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 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 81 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 [9]. Figure 11 reveals several cases of bug re-opening (such Solved-to-Inwork, Terminated-to-In Decision transitions).

VI. Empirical Analysis and Results

We conduct a series of experiments on real-world data from Siemens product lines to evaluate the effective of our approach. We conduct experiments on two projects to investigate the generalizability of our approach. One of the projects is a Image processing based product (Project A) deployed in Computed

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

	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 PROJECT B

	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

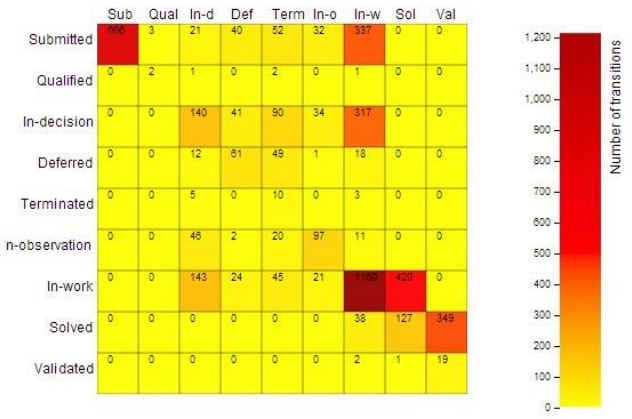

Fig. 11. A Heat Map showing the Number of Transitions (during the Bug Lifecycle of all Bug Reports in the Dataset) between the 8 Possible States

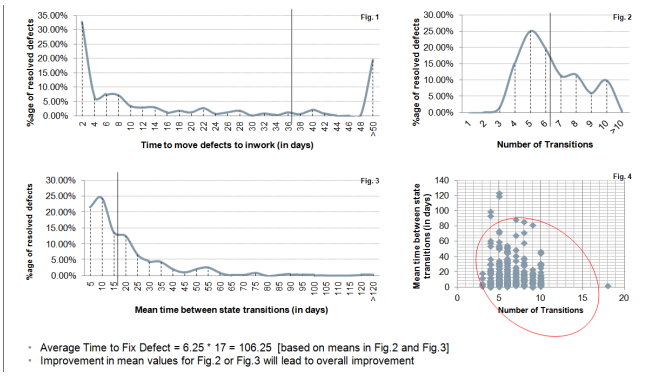

Fig. 12. Metrics to be used to track the effectiveness and usefulness of using the recommendation system.

Tomography Scan Machine. Project A started in 2012 and has 772 bugs reported till November 2014. Out of the 772 bug reports present in the Issue Tracking System, 345 have been solved and validated. We found that 236 issues have been closed due to either being duplicate bug reports or bug reports invalidated by the triager due to insufficient information to reproduce the bug. At the time of conducting the experiments, a total 78 members (project manager, product manager, testers, developers, test leads) are working on the project. The second project (Ultra-Sound Clinical Workflow Management System) is a relatively larger project (Project B) which started in 2005 and there are 17267 bugs reported till October 2014. Out of 17267 reported bugs, 12438 are resolved. A total of 253 professionals have worked on the project during the past 9 to 10 years. We consider only the resolved bugs for the purpose of conducting our experiments. N folds cross validation with N = 10 and K = [1, 10] is used for computing the precision and recall performance evaluation metrics. The formulae used for calculating precision@K and recall@K in information retrieval systems are as follows (where K is the number of developers in the ranked list):

$$Recall@K = \frac{1}{B}\sum_{i=1}^{B}\frac{|P_i \cap R_i|}{|R_i|}$$

$$Precision@K = \frac{1}{B}\sum_{i=1}^{B}\frac{|P_i \cap R_i|}{|P_i|}$$

In the formula for Precision and Recall, B denotes the number of bug reports, R represents the set of actual resolvers for a bug and P is the set of Predicted resolvers for the bug.

The calculated values have been shown in Tables I and II. We observe that the recall values increase as we increase K (which is quite intuitive). At K = 10, Project A has a recall of 0:734 with 345 solved bugs, whereas Project B has a recall of 0:905 with 1000 of the latest solved bugs. We observe that the precision values are maximum at K = 2 in both the projects. This is because in both projects the average number of resolvers per bug is very close to 2.

We conduct a manual analysis and visual inspection of a large number of bug reports and identify several instances in which a bug report is assigned to a bug fixer based on prior experience, workload, recent activity and severity and not just based on the closest match in terms of problem area expertise. We observe that in several cases the top recommended resolver (by our prediction model purely based on similar content-based recommendation) does not get the bug assigned due to factors such as workload and prior work experience of developers with the project incorporated in our decision support tool but not within the Decision Tree and

Naive Bayes based classification model. In one of the bug reports (status transition from in-work to in-work), we see a developers comments

- Due to workload issue, Alan was able to solve it partially and it needs to update the resolver.
- Since the bug is related to SRC Component, Todd has the experience in solving SRC related bugs and assigns the bug to Todd instead of Ramesh.
- The bug is high priority and assign it to Abhishek.
- Assign partial work to Rashmi and partial work to Manju
- Please assign this bug to me (I have been working in it recently).

Our manual inspection of several bug reports and the threaded discussions across two active projects in our organization demonstrates that factors in addition to content based assignment needs to be presented to the decision maker (as incorporated in our proposed decision-support system) for making better triaging decisions. In order to enable the projects to track the effectiveness and benefits of using the recommendation system, we proposed simple process metrics as shown in figure 12. The metrics are calculated for Project A, considering the data from the bugs that have already been resolved.

VII. 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 Naive Bayes learners and incorporating factors such as workload, prior work experience, recent activity and severity of bugs 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] A. Sureka, A. Goyal, and A. Rastogi, "Using social network analysis for mining collaboration data in a defect tracking system for risk and vulnerability analysis," in *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, (New York, NY, USA), pp. 195–204, ACM, 2011.

[7] 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.

[8] S. Lal and A. Sureka, "Comparison of seven bug report types: A case-study of google chrome browser project," in *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, vol. 1, pp. 517–526, Dec 2012.

[9] 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.