Collaborative Bug Triaging using Textual Similarities and Change Set Analysis

Katja Kevic, Sebastian C. Müller, Thomas Fritz, and Harald C. Gall
Department of Informatics
University of Zurich, Switzerland
katja.kevic@uzh.ch
{smueller, fritz, gall}@ifi.uzh.ch

***Abstract*—Bug triaging assigns a bug report, which is also known as a work item, an issue, a task or simply a bug, to the most appropriate software developer for fixing or implementing it. However, this task is tedious, time-consuming and error-prone if not supported by effective means. Current techniques either use information retrieval and machine learning to find the most similar bugs already fixed and recommend expert developers, or they analyze change information stemming from source code to propose expert bug solvers. Neither technique combines textual similarity with change set analysis and thereby exploits the potential of the interlinking between bug reports and change sets. In this paper, we present our approach to identify potential experts by identifying similar bug reports and analyzing the associated change sets. Studies have shown that effective bug triaging is done collaboratively in a meeting, as it requires the coordination of multiple individuals, the understanding of the project context and the understanding of the specific work practices. Therefore, we implemented our approach on a multi-touch table to allow multiple stakeholders to interact simultaneously in the bug triaging and to foster their collaboration. In the current stage of our experiments we have experienced that the expert recommendations are more specific and useful when the rationale behind the expert selection is also presented to the users.**

***Index Terms*—bug triaging; collaboration; expert recommendation; multi-touch;**

I. Introduction

Bug triaging is an essential part of developing software. Based on features of the bug report[1], such as the title, priority, severity, and affected components, developers have to assess whether a bug report is meaningful and identify a developer most suited for fixing the bug or implementing the required enhancement [1], [2]. This task of identifying potential experts for addressing bug reports is known to be time-consuming, tedious and error-prone, in particular due to the size and complexity of software projects and teams [3].

In the last few years, a variety of research approaches has been developed to automatically support bug triaging by recommending expert developers for bug reports. These approaches mainly differ in the way they identify the expert developers. The one kind of approaches focuses on textual similarity of the bug reports and bases on the assumption that bug reports that are similar in their textual characteristic should be fixed by the same developers. These approaches focus solely on discovering textual similarities between bug reports and use machine learning and information retrieval without looking at the code (*e.g.* [4], [5], [1]). The other kind of research has looked into identifying expert developers by using source code changes (*i.e.* change sets). These approaches are based on the assumption that the more changes a developer has made in the affected part of the code, the more of an expert she is in it (*e.g.* [6]). To determine an expert for a bug report, researchers then use information retrieval techniques to identify parts of the code that are related to a bug report and recommend the developers that are considered experts in these parts [7], [8].

Current approaches consider either bug report or change set information to determine a list of expert recommendations and do not take advantage of the links that exist between previously resolved bug reports and change sets. In contrast to the state of the art that mainly focuses on providing a list of recommendations, we pay attention to the collaborative nature of bug triaging [9] and provide more context for the recommendations.

In this paper, we present our approach to support collaborative bug triaging that combines information from bug reports with change sets. Similar to previous research, our approach uses an information retrieval method to identify bug reports similar to the bug being triaged. However, based on the identified bug reports, we then analyze the change sets associated with each bug report. Based on this analysis we then determine a refined and more qualified list of expert developers for addressing the bug. Furthermore, the associated change sets are used to provide a rationale for exploring the different recommendations and for supporting a more informed decision process. Finally, to facilitate the collaborative nature of bug triaging, our approach is targeted at an interactive user interface implemented as a prototype for the Microsoft Surface Table (MST).

In particular, this paper makes the following contributions:

- It introduces an approach for collaborative bug triaging that (a) identifies expert developers based on a combination of bug report and change set analysis, and (b) provides a rationale for choosing some recommended experts.

[1]In our approach, the term bug reports also encompasses tasks, issues and other work items.

- It presents our full prototype of the approach implemented on top of the Microsoft Team Foundation Server and the Microsoft Surface Table (MST) to support the collaboration of developers.

The remainder of this paper is structured as follows: Section 2 illustrates our approach with a usage scenario and presents details on the approach and its prototypical implementation. Section 3 provides a discussion of our approach and highlights future work. Related work is presented in Section 4 and we conclude our paper in Section 5.

II. Collaborative Bug Triaging

To best support developers with bug triaging, we developed an approach that extends research on expert recommendation and integrates it into a collaborative environment. Our approach reuses the idea that new bug reports might best be addressed by developers who previously addressed the bug reports that are textually most similar. Since there are often multiple people that work and submit changes for the same bug report [10], our approach further refines the ownership concept by analyzing all change sets associated with the previously addressed bug reports.

Studies have shown that triaging of new bugs requires the communication and collaboration between developers [10]. Also, it can be critical to find related bugs and understand the interdependencies between people's code in the process [9]. To address these aspects, our bug triaging approach does not only provide recommendations on expert developers but also information on related bugs and linked change sets. Furthermore, to better support the communication and collaboration between developers, we explicitly designed our approach for interactive tabletops that allow multiple people to interact with the information simultaneously. Other research has already shown that interactive tabletops inherently support collaboration (*e.g.*, [11]).

To examine its feasibility, we developed a prototypical implementation of our approach for the Microsoft Surface Table. This prototype is built on top of the Microsoft Team Foundation Server[2] that contains bug report, change set and people information in one integrated repository.

In the following, we will first illustrate our approach with a short usage scenario and then describe the approach and our prototype in more detail.

A. Usage Scenario

We introduce our approach by showing how several software developers can use our approach in a collaborative bug triaging session. Consider two software developers—Alice and John—that meet for assessing and assigning new incoming bug reports.

The developers start with our approach's *bug management perspective* (see Figure 1). Since Alice and John are only interested in new and unassigned bug reports from the current iteration, they quickly select the current iteration from the condensed iteration list on the right and then apply a filter using the filter action on the left of the perspective. This results in the two new and unassigned bug reports being displayed in the perspective.

After quickly skimming over the bug reports, they decide to first triage the bug report with ID 113 (bug 113). When Alice touches the *Analyze* button of the corresponding bug report on the MST (denoted by rectangle 1 in Figure 1), the *bug analysis perspective* opens (see Figure 2). This perspective displays a graph showing the bug reports that are textually similar to bug 113, in this case three, together with their similarity measure (see rectangle 1 in Figure 2). In addition, the approach shows the number of change sets associated with these bug reports (see rectangle 2 in Figure 2) and the developers who submitted these change sets and thus represent the set of recommended expert developers for addressing bug 113 (see rectangle 3 in Figure 2).

For each possible expert developer, Wendy, Tim and Tom in this scenario, an expertise score is presented that combines the bug similarity with the number of change sets. Based on these scores, Tim seems to be the best to address bug 113. However, after a quick investigation and discussion of Tim's change sets, Alice and John see that Tim only worked on layout issues not covering the work required for bug 113 in their opinion. Therefore, they look into Wendy and examine her change sets to the level of the actual code as illustrated in the lower part of Figure 2. After a short discussion, Alice and John come to the conclusion that Wendy is a good fit for addressing bug 113 and assign the bug to her. The assignment triggers our approach to send an automatic email notification to Wendy. Given that both, Alice and John, thought that her changes in the StartWindow.xaml.cs looked like a promising starting point for addressing the bug and they marked that on the user interface, the notification email already contains a reference to the code that Alice and John used in their decision process.

B. Approach

To support developers with bug triaging, our approach consists of four parts: the creation of a vector space for all existing bug reports on a project; the identification of bug reports similar to the bug report for triaging; the identification of expert developers based on a combined score of bug and change set analysis; and an interactive and collaborative environment that provides context for a more informed decision process. In the following, we describe each of these parts in more detail.

Creating the Vector Space for Bug Reports. Our approach is based on the assumption that for any unassigned bug, a developer who has previously addressed more textually similar bugs has a higher expertise than someone who has addressed less textually similar bugs. As a prerequisite for the identification of similar bug reports, our approach creates a vector space for all bug reports of a software project

[2]http://www.microsoft.com/visualstudio/deu/products/visual-studio-team-foundation-server-2012

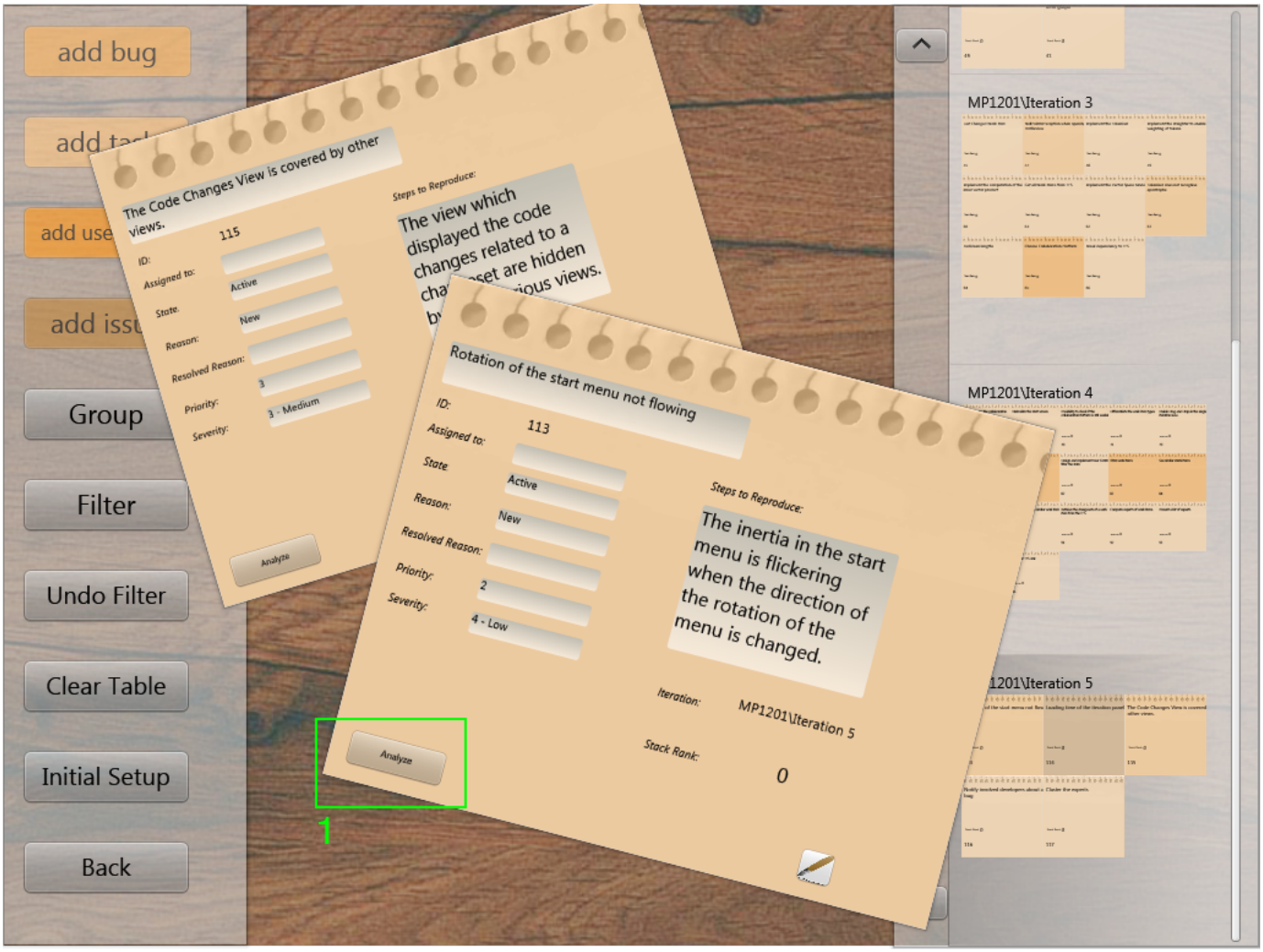

Fig. 1. The *bug management perspective* of our application that can be used to search for bug reports according to specific search criteria. The colored and numbered rectangle is not part of the user interface and denotes a part of the user interface that is described in the text in more detail.

after retrieving them from collaboration platforms such as the Microsoft Team Foundation Server or Rational Team Concert[3]. In particular, it uses a vector representation of free text and creates document vectors in a vector space for all retrieved bug reports from the software project of interest. To create the document vectors, we analyze the text in the title and in the description of each bug report and we apply tf-idf (term frequency - inverse document frequency) [12] as a weighting method. To improve the results later on for the identification of similar bug reports, we use text preprocessing activities to remove all stop words and punctuation in the text and build the stem of each word.

Identifying Similar Bug Reports. Given a bug report that is to be triaged, our approach extracts the information in the bug report into a query vector, also applying the already mentioned text preprocessing steps. Additionally, the words in the bug report are matched against the words already included in the vector space. If a word is not yet included in the vector space, its synonym is looked up[4] and used instead in case the synonym occurs in the vector space. To generate a ranked list of similar bug reports, our approach then computes the cosine similarity [13] between the query vector and each document vector in the vector space for the software project. A higher value indicates that two vectors are more similar and therefore the corresponding bug reports represented by these vectors are textually more similar. Our approach allows the user to adjust the similarity threshold for the inclusion of bug reports into the result set. This allows a user to adjust the risk of losing potential similar vector pairs because of a too high threshold, or the risk of including too many false positive results.

Identifying Expert Developers. To identify software developers with expert knowledge for an unassigned bug report, we first compute the list of similar bug reports. For each bug report in

[3]https://jazz.net/products/rational-team-concert/

[4]We use the NHunspell library for looking up synonyms (http://nhunspell.sourceforge.net/).

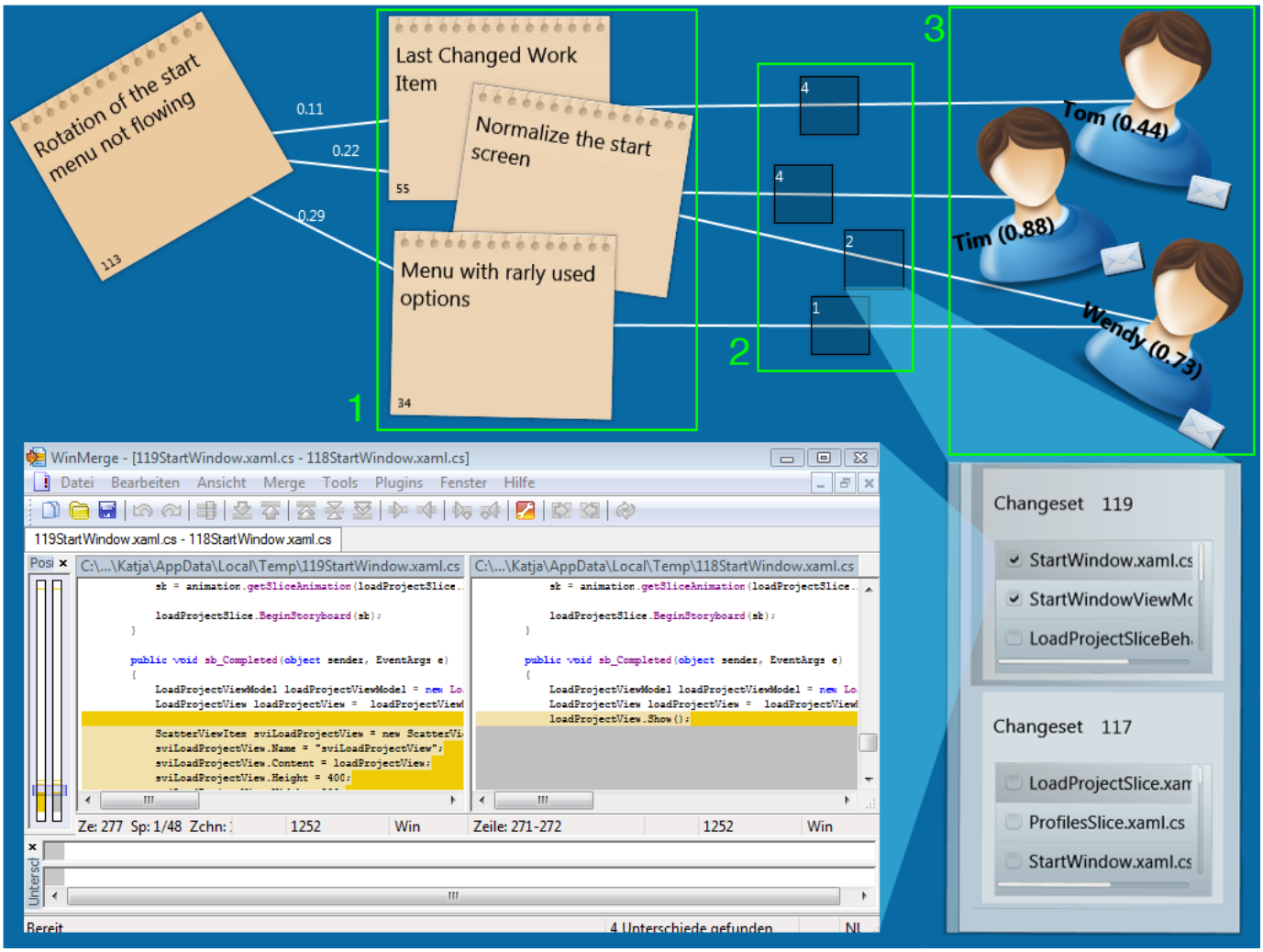

Fig. 2. The *bug analysis perspective* shows similar bug reports as well as suggested experts for a specific bug that is to be triaged. In this example, three similar bug reports were found for the analyzed bug and three software developer (Tim, Wendy and Tom) were recommended as potential experts. In the bottom right-hand corner, one can see two of the three change sets that were made by Wendy. For the first of these change sets, the view in the bottom right-hand corner shows the differences to the previous version.

this list, we then analyze the number of associated change sets and the developers who authored the change sets. Based on the assumption that a developer has a higher expertise for an unassigned bug report, if she delivered more change sets on similar bug reports, we compute an expertise score for each developer that authored a change set for one of the bug reports in the list. Specifically, this expertise score is calculated as the sum of all paths leading from the bug report to be triaged to a developer who authored a change set (see Figure 3). The value for each path is the cosine similarity multiplied by the number of change sets a developer authored for a similar bug report. In the example presented in Figure 3, our approach found three similar bug reports that exceeded the similarity threshold (in our case 0.3)[5], with a cosine similarity of 0.85, 0.78 and 0.71. The change sets that are associated with these three bug reports were committed by two different software engineers. Developer 1 authored seven change sets for bug report 1 and one for bug report 2, her expertise score is thus $7 * 0.85 + 1 * 0.78 = 6.73$. As a result of this step, we have a list of potential expert developers to address the bug report and a score representing the expertise of each of the developers.

Providing Context for a More Informed Decision in a Collaborative Environment. We aim to provide users of our approach with more context for understanding and selecting an expert recommendation. Therefore, our approach automatically re-

[5]In general, the similarity threshold should be set depending on the size of the bug database. For bug databases with a high number of reports, the threshold can be set higher, since it will be more likely to have similar bug reports than in smaller databases. We leave the determination of good thresholds for bug databases to future work.

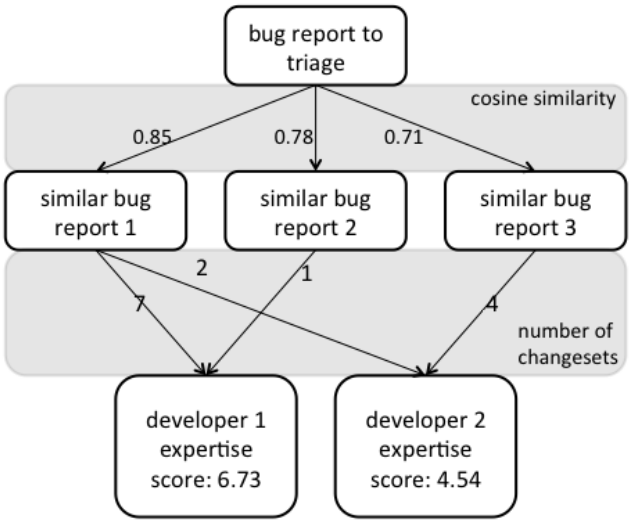

Fig. 3. An example of the approach our application uses to calculate the expertise score of potential experts. To calculate the expert level, similar bug reports and their associated change sets are considered, and a weighting method is applied.

trieves change sets that are associate with any of the similar bug reports and visualizes information on the bug reports, the similarity measure, the change sets and the potential expert developers together with their expertise scores in a graph representation as shown in Figure 2. The approach is developed for an interactive touch tabletop to ease and support collaboration between developers. Furthermore, it allows users to interactively explore each information artefact in more detail, such as the change sets (see Figure 2) or manage, select and filter bug reports as shown in Figure 1.

To provide an expert developer with a starting point for a newly assigned bug report, our prototype automatically tracks the code files inspected during bug triaging. In addition, users can manually remove or add files to the list of tracked files. At the end of the bug triaging process, our approach automatically sends this list to the newly assigned expert developer together with the bug report.

C. Prototypical Implementation

We developed a prototypical implementation of our approach as a proof of concept and for more detailed evaluations. The current version of our prototype[6] contains all features described in this paper, integrates with the Microsoft Team Foundation Server 2010, and runs on the Microsoft Surface Table 1.0.

The architecture of our prototype consists of five main components as shown in the architecture overview in Figure 4, an adapter for the collaboration platform, a data model, an engine to calculate similarity, a view model and an interactive user interface.

[6]The source code of the prototype is available at http://www.ifi.uzh.ch/seal/research/tools/collaborative-bug-triaging

The *Collaboration Platform Adapter* is responsible for retrieving bug reports and change sets from a collaboration platform, currently the Microsoft Team Foundation Server, and storing them in our data model. Our prototypical implementation has a *Work Item Service* that retrieves and automatically synchronizes bug reports with the server in the background on a regular basis and a *Version Control Service* to extract the change sets. Both services use APIs provided by the Microsoft Team Foundation Server. For our prototypical implementation, we focused on a collaboration platform that provides traceability features between bug reports and change sets. This means, that such a collaboration platform requires the software engineers to link at least one bug report to each change set they commit. However, our approach would generally also work with traditional repository mining techniques to recover the linkage between bug reports and change sets.

The *Data Model* represents the model underlying our approach that consists of bug reports, change sets and developers as well as the links between the elements in the model. For each element in the model as well as each link, the model captures the information relevant to compute expert recommendations and the information the user can explore interactively to better understand each computed recommendation.

The primary purpose of the *Similarity Engine* is to identify similar bug reports based on the description and the title of a given bug report. This component consists of a module that is able to preprocess text (*e.g.* to remove stop words and punctuation from the description or title of a bug report or to apply stemming), another module responsible for creating document and query vectors out of bug reports, and a module that calculates the cosine similarity between document vectors and a query vector.

The *View Model* contains the presentation logic and defines the bindings between data and its representation in the views of the user interface. It exposes data contained in the data model to the views of the user interface, ensuring that the architecture of the application obeys the MVVM pattern[7].

The *Interactive User Interface* is responsible for presenting all the data to the user and provides means to interact with the data intuitively and collaboratively. In particular, it contains all perspectives the user can interact with. Exemplary, two of them are shown in Figure 1 and Figure 2.

To better support the collaborative nature of bug triaging, our prototypical implementation provides various features that might help to foster the collaboration between users. For instance, almost each view in the user interface can be rotated, resized, and moved. This way, we provide an orientation-independent user interface that can be easily accessed by every user around the Microsoft Surface Table.

The prototypical implementation offers additional features that are not described in detail in this paper. For instance, we provide an action to group bug reports on the Microsoft Surface Table into issues, bugs, user stories and tasks or to

[7]http://msdn.microsoft.com/en-us/magazine/dd419663.aspx

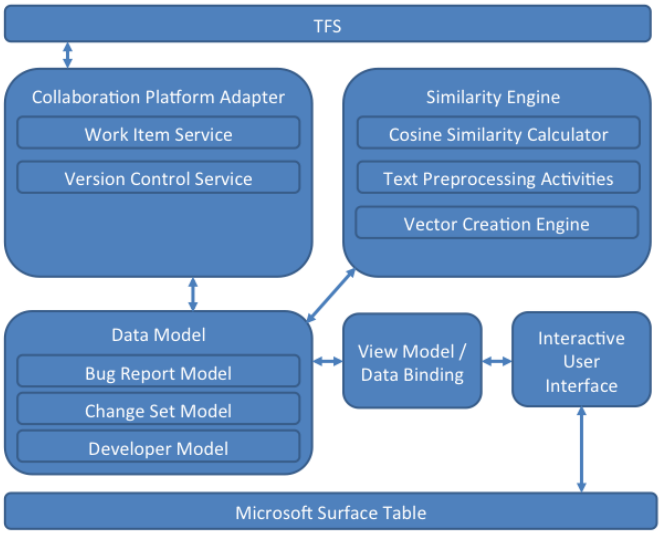

Fig. 4. Overview about the main components in the architecture of our prototypical implementation and the data flow between them.

remove all bug reports from the table by pressing the clear table button (see Figure 1).

III. Discussion and Future Work

We take advantage of the interlinking between bug reports and combine the information retrieval technique on bug reports with an analysis of associated change sets. This allows us to refine the recommendations of potential expert developers. Furthermore, it provides additional context for the decision process on who should address a bug and possibly where to start. We hypothesize that this combination leads to more accurate recommendations and an easier assessment of the recommendations. This additional context may also help in identifying good starting points for fixing a bug or to identify who could be affected and should stay aware of a bug.

Our approach currently focuses on a developer's previous activities or expertise for the bug triaging process. Other aspects, such as a developer's current availability and workload balance, might also be valuable factors for assignments in the bug triaging process. Therefore, we plan on further investigating these factors and how to best integrate them into the bug triaging process and the recommendations.

To support the collaborative aspect of bug triaging, we moved away from traditional point-and-click interfaces and developed our application for a multi-touch tabletop, since studies have shown that these inherently support collaboration [11]. By switching to a multi-touch environment, the application, as well as the bug triaging process itself, become more interactive and therefore might foster the collaboration between its users. To integrate the collaborative and interactive aspects into our application, we provide, besides other aspects, an orientation-independent user interface, that allows for rotating, resizing and moving almost each element of the user interface. Ideally, a user should be able to interact with our application as easy and intuitive as she interacts with real world content.

In contrast to traditional desktop solutions that constrain a user to sitting in front of a computer screen, tabletops have a form factor that allows for more natural face-to-face interactions with other users. Furthermore, on multi-touch tabletops, interaction with the content can take place simultaneously. In traditional approaches, one person is typically using the mouse or keyboard to interact with the user interface, while the other users stay passive. We hypothesize that our approach for bug triaging with a multi-touch tabletop environment will foster more interaction and collaboration between users in that process.

So far, we applied our approach only in our own software projects and experienced the usefulness of having additional context for the expert recommendations rather than just a simple list. As a next step, we plan to conduct user studies to evaluate our hypotheses, in particular the ability of the prototype to facilitate collaborative bug triaging.

Finally, we plan on further investigating our approach to provide an effective starting point to address a bug report. For now, our solution simply generates a list of all code elements from the change sets the users interacted with in the past. With a deeper analysis of all change sets that are associated with similar bug reports and their structural context, we might be able to automatically identify a smaller and more relevant set of code elements that serves as a good starting point for addressing a bug. At the same time, this set of code elements could also be helpful in the bug triaging process and provide more context for bug reports.

IV. Related Work

Work related to our approach can broadly be categorized into two areas: first, research focusing on detecting similar bug reports or analyzing code history for recommending potential domain experts and second, work that use multi-touch interfaces to support typical software development activities and foster the collaboration between the participants.

A lot of research has focused on identifying and recommending potential experts for fixing a bug or for components in a software project. In general, the research can be separated into two groups according to the technique used to recommend experts. The first group comprises approaches that mine and analyze source code repositories and source code history to identify experts. For instance, McDonald *et al.* [6] use a form of the "Line 10 Rule" in their Expertise Recommender, which implies that the person that last changed a code file has expertise in that file since it is freshest in her memory. Other approaches have extended these measures, for instance Kagdi *et al.* [14], who also analyze source code repositories and differentiate between two types of expertise, deep and wide expertise. Researchers also used heuristics on source code history to identify potential expert developers for bug reports, *e.g.*, Kagdi *et al.* and Linares-Vasquez *et al.* [8], [7] who first retrieve source code components related to a bug report using an information retrieval technique and then analyze the version control history of these components to recommend potential experts.

The other group of research approaches investigates textual similarities between bug reports to identify potential expert developers. For instance, Weiss *et al.* and Zhou *et al.* [4], [5] use vector-based methods while Anvik *et al.* and Liu *et al.* [1], [15] use supervised machine learning techniques. A similar approach is presented by Nagwani *et al.* [16]. They analyze the description of already assigned bug reports and generate a set of frequent terms for each bug report. These terms are then associated with the software developer to whom the bug report is assigned. Every time a new bug report has to be assigned, the set of terms of this bug report is matched against the terms associated with each available software developer. By ranking the matching results, Nagwani *et al.* are able to recommend potential experts. Jeong *et al.* [17] use a different approach. They analyze the reassigning of bugs to different software developers (*bug tossing*) and create a graph of these bug reassignments. A machine learning algorithm is then applied to reduce long bug reassignment paths. Jeong *et al.* argue that this approach can decrease the effort that is needed to find the software developer who should resolve the bug. In our approach, we use some of the techniques of these approaches as a basis and put it into an approach to foster collaboration and interaction.

There are only a few approaches investigating how multi-touch interfaces can be used to support typical software engineering activities. These approaches focus mostly on supporting agile planning meetings and on providing awareness of the current status of a software project but not on bug triaging. Wang *et al.* [18], for instance, describe their application that can be used in a agile project planning meeting to create, pass, toss, resize and rotate story cards. Morgan *et al.* [19] present a similar approach that supports both co-located as well as distributed teams in project planning meetings. This idea is augmented by Ghanam *et al.* [20] by adding an orientation-independent user interface for a multi-touch enabled application. Some approaches focus on other activities, such as Müller *et al.* [21] who present an approach to make code reviews more interactive and collaborative by putting them into a multi-touch environment. None of these approaches directly investigates the usage of multi-touch interfaces for bug triaging. Finally, there is one other approach we know of that focuses on improving the collaborative aspect of bug triaging. Bortis *et al.* [22] introduce their tool TeamBugs and suggest that it might support coordination among developers while bug triaging by providing extra feedback or guidance.

V. Conclusion

Bug triaging is an essential part in the software development process. Due to the complexity and size of software projects and teams, bug triaging can be very time-consuming, tedious and error-prone. In this paper, we have introduced an approach that automatically recommends potential experts for addressing unassigned bug report and provides context to ease the understanding of these recommendations. In contrast to previous research, our approach is based on a combination of determining textual similarities between bug reports and analyzing change sets to refine the recommendation results. Furthermore, it does not only present a list of recommendations to the user but provides information on the artifacts and the interlinking between them that triggered the recommendation to facilitate a more informed decision process. As a side effect, our approach might also be used to identify software developers who could be affected by a bug fix and provide a starting point for fixing a bug.

To support the collaborative aspect of bug triaging, we developed our approach for a multi-touch environment that is more interactive than traditional point-and-click interfaces. We implemented a prototype as a proof-of-concept and we plan a more thorough evaluation of our technology in a user study.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, (New York, NY, USA), pp. 361–370, ACM, 2006.

[2] C. R. Reis, R. P. de Mattos Fortes, R. Pontin, and M. Fortes, "An overview of the software engineering process and tools in the mozilla project," in *Proceedings of the Open Source Software Development Workshop*, pp. 155–175, 2002.

[3] O. Baysal, R. Holmes, and M. Godfrey, "Revisiting bug triage and resolution practices," in *User Evaluation for Software Engineering Researchers (USER), 2012*, pp. 29 –30, 2012.

[4] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, (Washington, DC, USA), IEEE Computer Society, 2007.

[5] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *34th International Conference on Software Engineering (ICSE)*, pp. 14 –24, 2012.

[6] D. W. McDonald and M. S. Ackerman, "Expertise recommender: a flexible recommendation system and architecture," in *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, CSCW '00, (New York, NY, USA), pp. 231–240, ACM, 2000.

[7] M. Linares-Vasquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pp. 451 –460, sept. 2012.

[8] H. H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *Journal of Software Maintenance*, vol. 24, no. 1, pp. 3–33, 2012.

[9] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg, "Designing task visualizations to support the coordination of work in software development," in *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, CSCW '06, (New York, NY, USA), pp. 39–48, ACM, 2006.

[10] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, CSCW '10, (New York, NY, USA), pp. 291–300, ACM, 2010.

[11] C. Shen, K. Ryall, C. Forlines, A. Esenther, F. D. Vernier, K. Everitt, M. Wu, D. Wigdor, M. R. Morris, M. Hancock, and E. Tse, "Informing the design of direct-touch tabletops," *IEEE Computer Graphics and Applications*, vol. 26, pp. 36–46, Sept. 2006.

[12] M. Lan, C. L. Tan, J. Su, and Y. Lu, "Supervised and traditional term weighting methods for automatic text categorization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, pp. 721 –735, April 2009.

[13] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[14] H. Kagdi, M. Hammad, and J. Maletic, "Who can help me with this source code change?," in *IEEE International Conference on Software Maintenance (ICSM)*, pp. 157 –166, 2008.

[15] K. Liu, H. B. K. Tan, and M. Chandramohan, "Has this bug been reported?," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, (New York, NY, USA), pp. 28:1–28:4, ACM, 2012.

[16] N. Nagwani and S. Verma, "Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes," in *9th International Conference on ICT and Knowledge Engineering (ICT Knowledge Engineering)*, pp. 113 –117, January 2011.

[17] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ESEC/FSE '09, (New York, NY, USA), pp. 111–120, ACM, 2009.

[18] X. Wang and F. Maurer, "Tabletop agileplanner: A tabletop-based project planning tool for agile software development teams," in *3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008.*, pp. 121 –128, October 2008.

[19] R. Morgan and F. Maurer, "Maseplanner: A card-based distributed planning tool for agile teams," in *Proceedings of the IEEE International Conference on Global Software Engineering*, ICGSE '06, (Washington, DC, USA), pp. 132–138, IEEE Computer Society, 2006.

[20] Y. Ghanam, X. Wang, and F. Maurer, "Utilizing digital tabletops in collocated agile planning meetings," in *Proceedings of the Agile 2008*, AGILE '08, (Washington, DC, USA), pp. 51–62, IEEE Computer Society, 2008.

[21] S. Müller, M. Würsch, T. Fritz, and H. C. Gall, "An approach for collaborative code reviews using multi-touch technology," in *CHASE*, pp. 93–99, 2012.

[22] G. Bortis and A. van der Hoek, "Teambugs: a collaborative bug tracking tool," in *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '11, (New York, NY, USA), pp. 69–71, ACM, 2011.