Which Bug Should I Fix: Helping New Developers Onboard a New Project

Jianguo Wang and Anita Sarma
Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, NE – 68588-0115
{jianguow, asarma}@cse.unl.edu

ABSTRACT

A typical entry point for new developers in an open source project is to contribute a bug fix. However, finding an appropriate bug and an appropriate fix for that bug requires a good understanding of the project, which is nontrivial. Here, we extend Tesseract – an interactive project exploration environment – to allow new developers to search over bug descriptions in a project to quickly identify and explore bugs of interest and their related resources. More specifically, we extended Tesseract with search capabilities that enable synonyms and similar-bugs search over bug descriptions in a bug repository. The goal is to enable users to identify bugs of interest, resources related to that bug, (e.g., related files, contributing developers, communication records), and visually explore the appropriate socio-technical dependencies for the selected bug in an interactive manner. Here we present our search extension to Tesseract.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques: User Interfaces

General Terms

Management, Human Factors.

Keywords

Socio-technical dependency, bug search, new developer.

1. INTRODUCTION

A big challenge in software development is to get developers to quickly become familiar with the project and its resources when they are assigned to a new project. Gaining an understanding of the relevant resources (e.g., project dependencies, technical shortcuts, social cultures) in a new project is vital for a new comer's successful integration into the project. However, this is often nontrivial and it takes significant time and effort before developers become familiar with the project features and overcome technical and social obstacles [5]. This problem of getting new developers acclimatized to their new project exists in both commercial [5] and open source projects [7].

A typical starting point for new developers in becoming familiar is to start by exploring the bugs and issues logged in a project. This is especially true for open source projects. In fact, Rhythmbox[10], a popular Gnome [6] project has the following recommendation for new developers: "*If you don't know what to work on, or you're looking for a small task to get started, take a look at the list of 'gnome-love' bugs and the current GNOME goals. Otherwise, there are enough bugs and feature requests in Bugzilla to keep anyone busy*."

However, new developers have difficulty finding the bugs that are of interest, that match their skill sets, are not duplicates, and are important for their future community. Typically, this phase takes significant time and effort since the developer has to manually trawl through the bug lists in the project's bug repository and/or issue tracker. Further, after the developer has identified a set of bugs that she wants to fix she has to then understand the different resources that are related to the bug. For example, for the new developer who is attempting to start on the bug fix it would be beneficial for her to know: whether a similar bug had occurred in the past, whether this bug could be related to a past feature fix, which other files are dependent on the file that is causing the bug, which other developers have worked on these files or features in the past, and so on. As we can see fixing a bug often requires a good understanding of both the social and technical dependencies in a project and tool support in helping a new developer navigate this complex project landscape will be beneficial.

In the recent past, enabling the efficient use of bug databases has gained attention and a few 'bug search' engines have been developed. Linkster [3] allows a developer to manually link bugs to its related bug-fix commits, which helps in overcoming some of the missing context that is very common in open source projects. However, this requires an experienced developer to post hoc create the links. InfoZilla [4] automatically extracts structural information from bug reports such as stack trace and source code, which provides a richer context to users thereby helping them with their bug fixes. DebugAdvisor [2] allows users to search related information for a bug in variegated repositories by creating a query that includes both structured and unstructured data describing the contextual information about the bug. Using probabilistic inferences it then recommends a ranked list of people, files, and other resources related to the bug. This work is closest to ours, but lacks the interactive exploration features we provide. These tools largely focus on helping current developers find related bugs in a repository by providing a better context for bugs.

Our focus, on the other hand, is in helping new developers to understand the complex socio-technical dependencies involved in fixing a particular bug and in helping answer some of the questions that a developer has before beginning to fix a bug in a new project. To do so, we enable users to search for a bug or feature and identify related bugs, relevant resources, and their dependencies in an interactive and graphical manner. We extended Tesser-

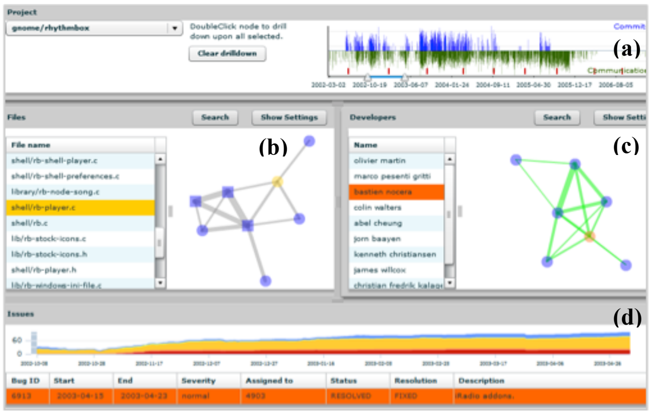


Fig 1. Tesseract Screenshot.

act [11], an interactive project exploration environment, with bug search capabilities by integrating a search engine that incorporates synonym expansion and document similarity search.

The synonyms search helps identify a larger set of bugs that might be of interest to developers by querying the search terms as well as their synonyms. The similar-bugs search recommends relevant bugs to a specific bug by searching for bugs that have similar bug descriptions in the repository. Additionally, unlike other search tools we support interactive visual exploration of the search results. More specifically, for each bug in the search result we present the relations between bugs and other resources such as files, developers, and communication records and allow the exploration of these resources in an interactive and visual manner.

The rest of the paper is organized as follows. In Section 2 we briefly introduce Tesseract followed by our approach in extending Tesseract to include bug search. We provide the implementation details in Section 3 and conclude in Section 4.

2. APPROACH

2.1 Tesseract

Tesseract is an interactive, visual project exploration environment that focuses on socio-technical dependencies in software projects [11]. It analyzes information from code archives, communication records, and bug repositories to capture the relations between code, developers, software bugs, and communication records. Key features of Tesseract include: (1) cross-linked panes that allow interactive exploration of technical relationships among files, socio-technical relations between files and developers, and social networks among developers; (2) highlighting of related files, developers and bugs; and (3) visualization of socio-technical congruence, i.e., the match between developers who are supposed to communicate because of underlying technical dependencies and developers who are actually communicating.

Tesseract consists of four panes (see Fig. 1). The top pane is a Project activity pane (Fig. 1-a), which displays the overall activities in the selected project – frequency of commits at the top and frequency of communication at the bottom. Note that the time period selected from this pane is reflected across all other panes.

The Files network pane (Fig. 1-b) displays relationships among files or packages in the project. Files are represented as round nodes and packages as square nodes. Files are considered interrelated and graphed as a network if they are committed together (user can select a desired threshold). Further, users can explore a package by drilling down (double clicking) on a package node, which expands it to the file level. A textual listing of file names is provided on the left for quick identification of artifacts.

The Developers network pane (Fig. 1-c) displays relationships between developers. Developers are considered to have technical dependencies if they either edited the same or interdependent files. Social dependencies are deciphered based on communications among developers through mail messages or Bugzilla activities. User names are displayed to the left for easy identification.

The Issues pane (Fig. 1-d) displays bug information for a selected time period. It consists of an area chart of open bugs and their severities, and a textual description of bugs and a search UI.

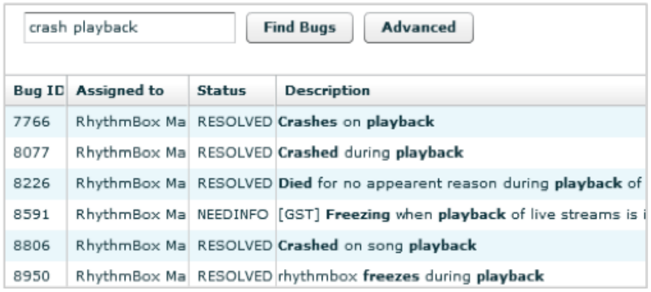
Finally, all four panes are cross-linked based on the underlying socio-technical relationships (e.g., developer who commits a file, or is assigned to a bug, or communicated about a bug) to aid the exploration of these relationships (see highlighted nodes in Fig. 1)

2.2 Integrating Search in Tesseract

A good entry point for developers into a project is to contribute a bug fix on a topic in which they are interested. However, doing so is nontrivial. When a developer chooses a bug from a bug repository or is assigned one in a project with which she is unfamiliar, she might spend significant effort and time searching (in an ad hoc manner) for past instances of bugs and relevant resources for that bug [2]. Automated support that helps developers find related bugs to a particular bug (or a feature) along with relevant resources for fixing that bug and their dependencies can help a new developer to quickly become familiar with a project.

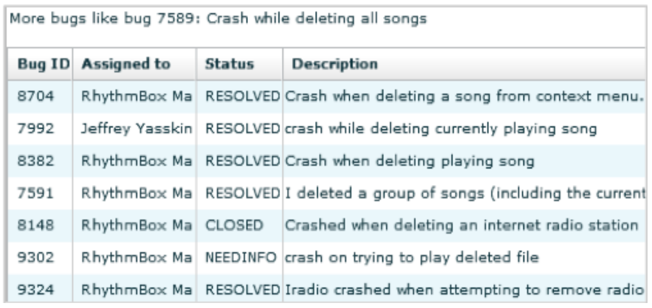
To achieve this goal, we extended Tesseract with a bug search feature to help new developers identify appropriate bugs and related resources, which can help them in becoming familiar with a project and get a jumpstart in making their contributions. In addition to regular query terms for bugs, our smart search incorporates synonym expansion of keywords and document similarity search on bug descriptions to recommend a larger set of relevant bugs, which are then ordered based on their closeness to the original query terms or a selected bug.

Synonyms Search: In addition to a regular keyword search query, we provide a *synonyms search* function to identify other relevant bugs that are similar, but do not contain the exact phrasing as the current bug. For example, when a user searches for “playback crashes”, the search results list all bugs with descriptions that contain the words “crash” and “playback” as well as “freeze” and “die”, which are synonyms to “crash” (see Fig. 2). The synonyms search can thus provide a larger set of bugs that might be of interest to the user than searching for only the exact words that the user queried. This also allows users to explore bugs that are relat-



Bug ID	Assigned to	Status	Description
7766	RhythmBox Ma	RESOLVED	**Crashes** on **playback**
8077	RhythmBox Ma	RESOLVED	**Crashed** during **playback**
8226	RhythmBox Ma	RESOLVED	**Died** for no appearent reason during **playback** of
8591	RhythmBox Ma	NEEDINFO	[GST] **Freezing** when **playback** of live streams is i
8806	RhythmBox Ma	RESOLVED	**Crashed** on song **playback**
8950	RhythmBox Ma	RESOLVED	rhythmbox **freezes** during **playback**

Fig 2. ‘Find Bugs’ tab on querying "crash playback".


More bugs like bug 7589: Crash while deleting all songs

Bug ID	Assigned to	Status	Description
8704	RhythmBox Ma	RESOLVED	Crash when deleting a song from context menu.
7992	Jeffrey Yasskin	RESOLVED	crash while deleting currently playing song
8382	RhythmBox Ma	RESOLVED	Crash when deleting playing song
7591	RhythmBox Ma	RESOLVED	I deleted a group of songs (including the current
8148	RhythmBox Ma	CLOSED	Crashed when deleting an internet radio station
9302	RhythmBox Ma	NEEDINFO	crash on trying to play deleted file
9324	RhythmBox Ma	RESOLVED	Iradio crashed when attempting to remove radio

Fig 3. 'Similar Bugs' tab on searching bugs similar to bz 7589.

ed to a feature by searching for terms in the feature or its name.

Similarity Search: Another search feature that we include in Tesseract is a *similar bugs search*. That is, when given a specific bug, the search engine identifies keywords from the bug description and searches for bugs with similar descriptions. For example, when a user is exploring a bug (bz_7859) in the repository and wants to identify other similar bugs, the search feature provides a list of bugs with descriptions that contain words that were in the description of bz_7859. In our example, the key terms in bz_7859 description were: crash, delete, and song. The search includes these keywords and their synonyms to retrieve the related bugs. The resultant set is then ordered based on the closeness of the results to the original bug (see Fig. 3). From this list, users can determine which bug(s) is interesting and continue to explore other similar bugs by drilling down on any of the results.

A key feature of our search is that the resultant bugs are reported and visualized in Tesseract, which allows users to identify the developers or files that were involved with a particular bug, developers who communicated regarding a particular bug, and so on. Fig. 1 shows the related files and developers of a bug that has been selected by a user.

3. IMPLEMENTATION

3.1 Background on Information Retrieval

Synonym expansion and document similarity search are two information retrieval techniques that we use. Synonym expansion is the process of expanding a word to its variants at either query or indexing time. In our case, we index a word in a document along with it synonyms if the word exists in our dictionary of synonyms. Document similarity uses different heuristics to retrieve a ranked listing of documents that are similar to a query document [12]. In our implementation, we follow the Cosine measure based on the vector space model [9] to retrieve similar documents. Documents and queries are modeled as n-dimensional elements of a vector space *$(w_1, w_2 \ldots w_n)$*, with n being the number of index terms and w_i reflecting the importance of each term *i* in document or query. As noted in Equation 1, term weight w_i is calculated as the product of term frequency (tf_i) and inverse document frequency (idf_i).

$$w_i = tf_i \times idf_i \qquad (Eq.1)$$

idf_i is calculated as in Equation 2 with D referring to the total number of documents and df_i being the number of documents with the occurrence of index term i.

$$idf_i = \log(D/df_i) \qquad (Eq.2)$$

Using the vector representation of documents and queries, the Cosine similarity between a query and an indexed document is calculated based on Equation 3.

$$Sim(q,d) = \frac{\sum_{i=1}^{n} w_{qi} \times w_{di}}{\sqrt{\sum_{i=1}^{n} {w_{qi}}^2 \times \sum_{i=1}^{n} {w_{di}}^2}} \qquad (Eq.3)$$

3.2 Implementing Search in Tesseract

We built our search features using Solr – an open source search platform [1]. The search feature comprises five phases: analyzing, indexing, searching, querying, and reporting (see Fig. 4). In the analyzing phase, the search engine retrieves bug data from the Bugzilla database, analyzes and preprocesses the data by stemming, filtering out stop words, and synonyms expansion. Stemming is the process for reducing words to their root. For example, "worked", "working" can both be stemmed to "work". Filtering out stop words refers to filtering out words such as "and", "a", "the", etc that provide little lexical meaning to improve performance. Synonyms expansion is explained later in this section. The analysis phase consists of first parsing the descriptive text of a bug into a bag of words (an unordered collection of words). In the indexing phase, the bag of words corresponding to a specific bug is indexed as a distinct document, which is then used in the search phase. Note that synonyms for each term in the bag of words are retrieved from our synonym thesaurus and indexed. In the query phase users can either query by key terms of a software feature or search for a similar bug by selecting a specific bug from the UI. In the reporting phase, search results are displayed based on a ranking of their closeness to the search query.

An important component of our search feature is the synonyms search. The first step for synonyms search was to create a synonyms thesaurus. We implemented it by manually analyzing bug descriptions to determine synonyms, which were then added to the thesaurus. Following are examples of synonyms in our thesaurus.

- *mute, silent*
- *view, display*
- *crash, freeze, die, not working, doesn't work*
- *delete, remove*

Manually creating the thesaurus was time consuming, but it is a resource that can be reused for other software engineering projects. It took one of the authors about 20 hours to analyze 2288 bug records in a project to create 100 synonym entries.

The next step in the process is synonym expansion, which is performed during the indexing process. That is, when a term generat-

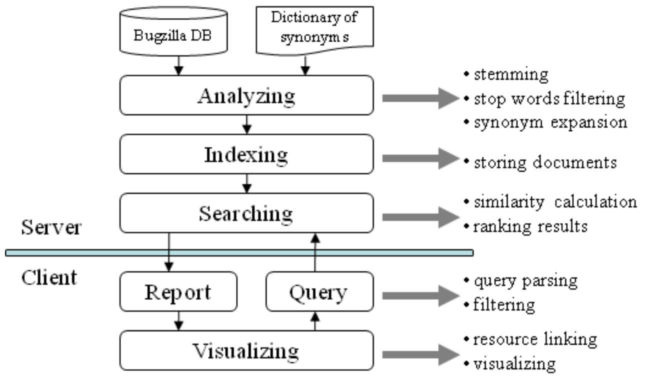


Figure 4. Steps in Search Engine

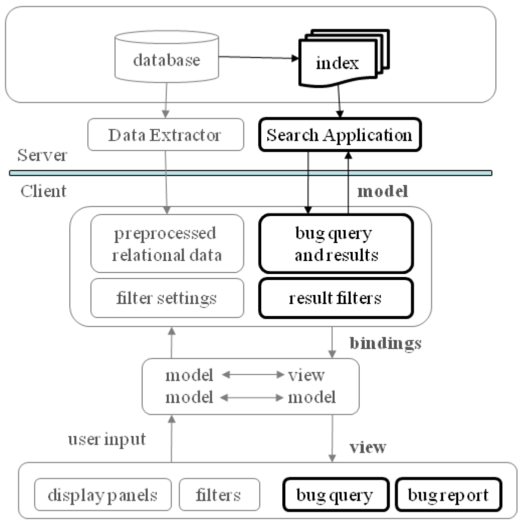


Figure 5. Integration of bug search in Tesseract.

ed from a bug description is being indexed the analyzer first checks whether the term has any synonyms in the thesaurus. If it does then all its synonyms are also indexed as terms. Because of this, when a search is performed all documents that contain the query terms or their synonyms are retrieved. Once a set of query results have been obtained they are ranked based on the closeness of their (bug) descriptions to the original query. More specifically, we use Cosine similarity (see Equation 3) to identify the similarity between the bug description and the query and rank the resultant bug reports accordingly.

The similar bugs search follows a comparable approach. The main difference being that a user selects a specific bug from the bugs pane list (see Fig. 1-d) instead of searching over specific query terms. When a user selects a particular bug, the search engine retrieves the description of the selected bug from the Bugzilla database using the bug ID. It then parses the description and converts it into a bag of words, which are considered as the key query terms for the search. The rest of the process is exactly the same as above. Once the search engine recommends the “similar” bugs, users can investigate the resultant bugs as well as explore other similar bugs from the result set.

Implementing the search feature required us to make modifications to the original Tesseract architecture. Fig. 5 shows the additions to the original architecture through the highlighted modules. Specifically, the search application server and index was added at the backend, new models on bug query and filters added, and two new UI components implemented (bug query, bug report).

The interactive project exploration feature provided by Tesseract sets us apart from other existing bug search tools. Using our search feature new developers can visually explore a particular feature, code component, or a bug. They can now easily explore the vast project space to identify other related bugs or feature fixes, the files and developers that were connected to a bug (or related bugs), developers who discussed about a bug and have the required expertise, and so on.

4. CONCLUSIONS

We extended Tesseract to incorporate search feature for bugs or issues in a project so as to help new developers familiarize themselves with their new project. That is, new developers can search for key terms of a feature or an existing bug and explore other related bugs and artifacts associated with that bug to get a better insight into the project. We incorporated synonyms expansion and document similarity in our search feature by building on Solr, an open source search platform [1]. The Tesseract search feature uses a synonyms thesaurus to also include synonyms of queried terms, which provides a larger set of related bugs to developers. Finally, we use Cosine document similarity measures to rank the resulting set of bugs. Users can now explore these bugs (and other similar bugs) through the cross linked views of Tesseract to explore the different dependencies in resources (files, developers, communication records) associated with the bugs.

Currently, we have created our own synonyms thesaurus. In the future, we will explore integrating external synonym resources like WordNet (a large lexical database of English) [8] or using machine learning techniques to automatically identify synonyms. We also plan to use natural language processing techniques like latent semantic analysis for more refined searches. Finally, we plan to conduct user studies to evaluate the usability and effectiveness of Tesseract and its search feature.

5. ACKNOWLEDGMENTS

This research is supported by grants NSF CCF-1016134 and AFSOR FA9550-09-1-0129.

6. REFERENCES

[1] Apache.org. Solr – an open source enterprise search platform. Available: http://lucene.apache.org/solr/

[2] B. Ashok, et al., "DebugAdvisor: A Recommender System for Debugging," in European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, 2009, pp. 373-382.

[3] A. Bachmann, et al., "The Missing Links: Bugs and Bug-fix Commits," in Foundations of Software Engineering, 2010, pp. 97-106.

[4] N. Bettenburg, et al., "Extracting Structural Information from Bug Reports," International Working Conference on Mining Software Repositories, 2008, pp. 27-30.

[5] B. Dagenais, et al., "Moving into a new software project landscape," International Conference on Software Engineering, 2010, pp. 275-284.

[6] GNOME. The Free Software Desktop Project. Available: http://www.gnome.org/

[7] G. v. Krogh, et al., "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study," Elsevier - Research Policy, 32(7), 2003, pp. 1217-1241.

[8] Princeton. WordNet - a large lexical database of English synonyms. Available: http://wordnet.princeton.edu/

[9] V. V. Raghavan and S. K. M. Wong, "A Critical Analysis of Vector Space model for Information Retrieval," Journal of the American Society for Information Science, 37(5), 1986, pp. 279-287.

[10] RhythmBox. Music Management Application for Gnome. Available: http://projects.gnome.org/rhythmbox/

[11] A. Sarma, et al., "Tesseract: Interactive Visual Exploration of Socio-Technical Relationships in Software Development," in International Conference on Software Engineering, 2009, pp. 23-33.

[12] X. Wan, et al., "Towards a unified approach to document similarity search using manifold-ranking of blocks," Information Processing & Management, 44(3), 2008, pp. 1032-1048