2009 IEEE International Advance Computing Conference (IACC 2009)
Patiala, India, 6–7 March 2009

Bug Mining Model Based on Event-Component Similarity to Discover Similar and Duplicate GUI Bugs

Naresh Kumar Nagwani, Pradeep Singh
Department of Computer Sci. & Engg.
National Institute of Technology, Raipur, India
nknagwani.cs@nitrr.ac.in , psingh.cs@nitrr.ac.in

Abstract- **All most all of the bugs related to Graphical User Interface (GUI) module of applications and are described in terms of events associated with GUI components. In this paper, a bug mining model for discovering duplicate and similar GUI bugs is presented and approach for detecting the similar and duplicate GUI bugs is described. Resolution of similar and duplicate bugs are almost identical, so if similar and duplicates are identified it will optimize the time for fixing reported GUI bugs and it can also help in achieving the faster development. A GUI bug can be transformed into a sequence of events, components and expected implementation requirements for each GUI event. This transformation is used in this paper to discover the similar and duplicate GUI bugs. First all the GUI bugs are transformed into events, components and requirements sequence, then these sequences are pair wise matched and common subsequence is generated which will indicate the similarity for the GUI bugs.**

Keywords- similar bugs, duplicate bugs, event component similarity;

I. INTRODUCTION

Bugs are defects in applications or in software's. During development phase of any application some of the implemented requirements do not meet the expected requirements, and during testing phase they are identified as a bug. Generally bugs are identified by a testers or a quality engineer in order to make sure that the implemented programs meets the given requirements. There are number of open source as well as commercial tool are available for bug tracking, some of them are JIRA, BugZilla etc. All the logged bugs are stored in bug database for tracking by these tools. All the bugs follow some standard path once they come in existence, which is known as the state flow of a bug. The state flow diagram of a bug is presented in Figure 1.

Here is one of the case of bug flow diagram is explained. When a new bug will enter into the system it is in new state. When developer will start working for this bug it will move to the in-progress state, if this bug is fixed by developer then developer will mark it as resolved and bug will enter into the resolved state. Once tester will verify this resolved bug if it is fixed then it will be in closed state.

Most of the applications today have a very good Graphical User Interface (GUI). GUI interfaces simplifies the process of taking input from user, displaying results in better ways and represents the application in decorated manners. But apart from this advantages side there is a side effect of GUI application, which is the development effort, involved in implementing a good GUI application is very expensive and complex, since developer has to keep in mind about the various actions performed by the user on the application's GUI. And there are possibilities that some of the actions can be missed out during documentation or development phase of the application, which results in the form of a bug. In this paper GUI bugs are targeted for similarity behaviour, if similar or duplicate GUI bugs can be discovered from the bug database then resolution time and effort can be optimized in better ways. One good thing about the GUI bugs is these are described in terms of GUI events (actions) and GUI components on which the events are fired. So using these properties of GUI bugs an approach is suggested here to discover similar and duplicate GUI bugs in a GUI bug database.

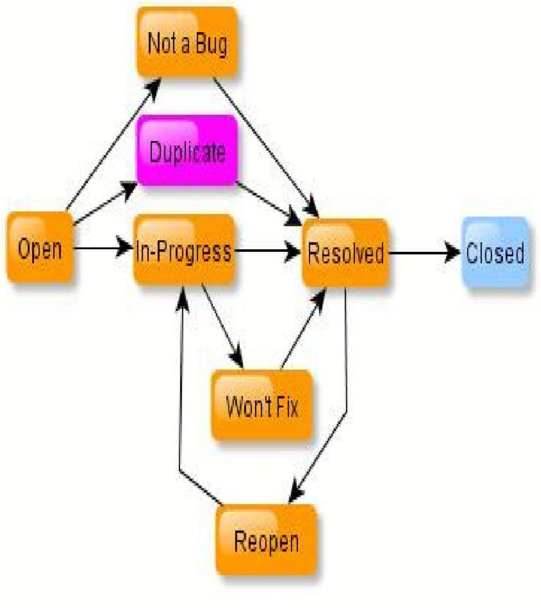

Figure 1. Bug State Flow Diagram

1.1 Similar and Duplicate Bug

In this paper two terminologies are used, one is similar bug and second is duplicate bugs. The basic difference between these two terminologies is, a bug is said to be similar if same implementation behaviour required for fixing two different reported bugs. And a bug is said to be duplicate if the same bug is reported more than once in different way, i.e. fixing one bug will automatically fix the second bug since the second one was the same defect which was mentioned in different way.

If two bugs are similar then fixing one bug will not be fixing another one but the solution of first bug fixation will be similar to solution to second bug fixation.

1.1.1 Similar Bug Example

Similar bugs are explained here with the help of examples. Assume that there are two assigned bugs:

Bug #1: Summary: Clicking on the close button should show a popup dialog to user, for confirmation.

Description: Whenever a user will click on the close button one popup will appear with the text "Do you want to exit?", on click yes application will exit on clicking no, popup will simply disappear.

Bug #2: Summary: Selecting File menu's exit option will show a popup dialog for exit confirmation.

Description: Whenever a user will select the exit option from file menu, one popup will appear with the text "Do you want to exit?", on click yes application will exit on clicking no, popup will simply disappear.

Finding similar bugs: For bug #1 and #2 the same implementation behaviour required for two different operations i.e. on close button click and on file menu exit click. So the same implementation is required for these two different events.

1.1.2 Duplicate Bug Example

Duplicate bugs are explained in this section with the help of examples. Now assume that there are two more bugs given below for analysis.

Bug #3: Summary: Clicking on the close button and file menu exit option should show a popup dialog to user, for confirmation.

Description: Whenever a user will click on the close button one popup will appear with the text "Do you want to exit?", on click yes application will exit on clicking no, popup will simply disappear.

Bug #4: Summary: performing application exit operation, application will show a popup dialog for exit confirmation.

Description: Whenever a user will select the exit option from file menu or clicking on close button, one popup will appear with the text "Do you want to exit?", on click yes application will exit on clicking no, popup will simply disappear.

Finding duplicate bugs: bug #3 and #4 are the same bug is reported in two different ways. If bug #3 will be resolved or fixed then #4 will also be automatically fixed, which means the bugs are duplicate.

Now the interesting point here is looking at which part of a reported bug it can be said that the reported GUI bugs are similar or duplicate to another. Generally most of the GUI bugs are reported is terms of summary or title of the bug, which comprises of GUI events or the actions by which the expected behaviour can be achieved, the GUI components on which the GUI events will be working and the expected requirement or behaviour which is desired once a event is applied on some GUI component.

Section 2 describes the work related to discovering the duplicate bugs in a bug database. In section 3 proposed approaches is explained, and in section 4 illustrative examples for the given approach is explained. And in section 5 conclusion and future directions for the paper is mentioned.

II. RELATED WORK

There are number of ways suggested by the researchers to discover the duplicate bugs in a bug database and some of the recent and popular are mentioned in this section.

The group of 5 persons Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik and Jiasu Sun has proposed discovering semantically similar terms using WordNet. Several methods have been implemented and evaluated. Also they have proposed the Semantic Similarity Retrieval Model (SSRM), a general document similarity and information retrieval method suitable for retrieval in conventional document collections and the Web. The basic concept which they have suggested was when a new bug report arrives; its natural language information and execution information are compared with those of the existing bug reports to discover the duplicate bugs [11].

The term-based Vector Space Model (VSM) (the state-of-the-art document retrieval method) and SSRM have been implemented and evaluated on a retrieval system for images and documents on the Web was given by Giannis Varelas, Epimenidis Voutsakis, Paraskevi Raftopoulou [3].

Shuji Morisaki, Akito Monden, Tomoko Matsumura, Haruaki Tamada and Ken-ichi Matsumoto have proposed an extended association rule mining method based on quantitative variables to provide an empirical study to reveal rules associated with defect correction effort. [10]

Nicholas Jalbert and Westley Weimer have proposed a system that automatically classifies duplicate bug reports as they arrive to save developer time. This system uses surface features, textual semantics, and graph clustering to predict duplicate status. [6]

2.1 Problem Identification

Although there are number of available approaches mentioned above to detect the duplicate bugs in bug database and some of the studies of duplicate record elimination are given but none of them discussed approaches for similarities of bugs in bug database and none of the approach suggest a GUI bug data representation techniques by which the operations for discovering the similar and duplicate bugs can be simplified. GUI bugs can easily be described in terms of GUI events, GUI components and requirements of expected behaviour. Using event-component-requirement structure presentation of GUI bug, process of discovering the similar and duplicate GUI bugs can be simplified.

III THE APPROACH

In this paper an event, component and requirement similarity based way of identifying the duplicate GUI bugs is proposed. First a GUI bug description is transformed into sequences of GUI events, GUI components on which the events are working and sequences of desired requirements based on the events. Once all the bugs are converted into these three sequences then pair wise these sequences are matched for the GUI bugs and longest matching subsequence is captured. This is the similarity between the GUI bugs. If the two event-component-requirements for two GUI bugs are exactly the same then these GUI bugs are said to be as duplicate bugs. Figure 2 depicts a bug mining model for discovering the similar and duplicate bugs. All the GUI bugs are converted into Event-Component structure, then similarity matching logic is applied to measure the GUI bug similarities and finally duplicate and similar bugs are identified.

3.1 GUI Bug Transformations into Event-Component-Requirement Sequences

Figure 3 depict the model for transforming a GUI bug into sequences of events, components and requirements. This model is divided into three parts – Input (GUI Bugs) module, Event Component (EC) Mapping module and Output (sequences of events, components and requirements) module. The EC Mapping module is again divided into three parts – Events Sequence Mapping (ESM), Components Sequence Mapping (CSM) and Requirements Sequence Mapping (RSM).

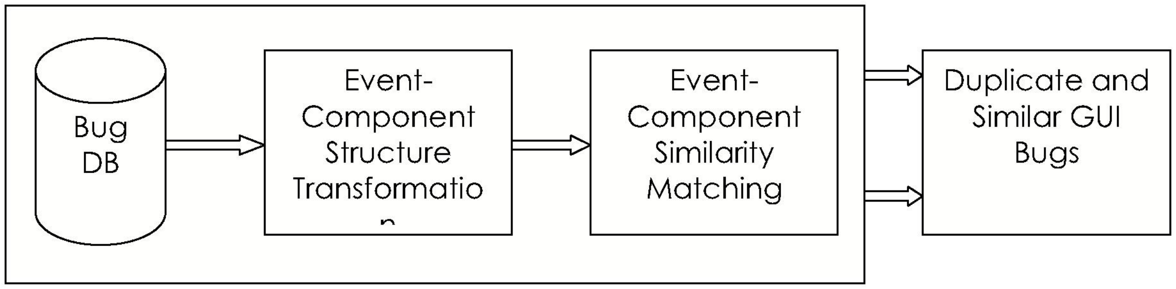
Bug DB
Event-Component Structure Transformation
Event-Component Similarity Matching
Duplicate and Similar GUI Bugs

Figure 2. Bug mining model to discover similar and duplicate bugs.

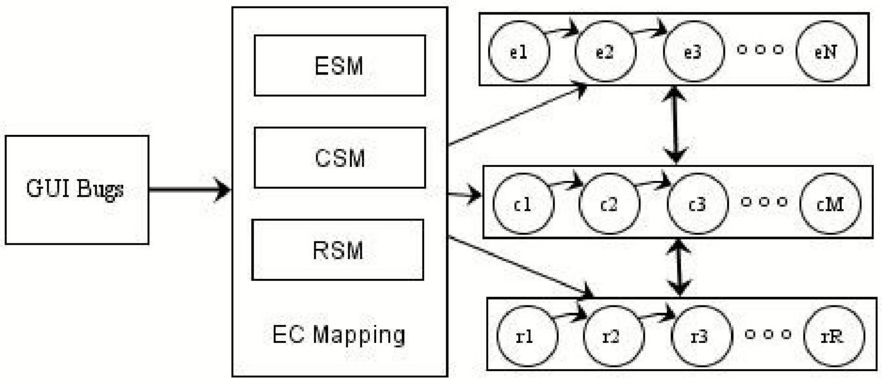

Figure 3. Transforming the GUI bug into the sequences of Events, Components and Requirements.

3.2 Event-Component-Requirement Sequence Similarities

Figure 4 represents the algorithm for matching longest similar subsequence of two event sequences for two GUI bugs. Event sequences for two GUI bugs are feed into the matcher, which will result in longest matching subsequence between these two event sequences. In Figure 3 two GUI bug event sequences are feed first bug consisting of N GUI events and second bug consisting of M number of GUI events, which results in I common event sequence for the given two bug's event sequences. In the similar manners Figure 5 and Figure 6 depicts the longest subsequence findings for components and requirements similarity matching.

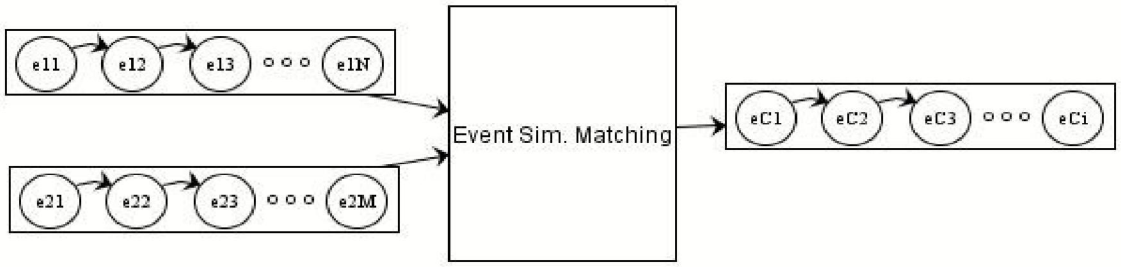

Figure 4. Events Sequence Similarity Matcher.

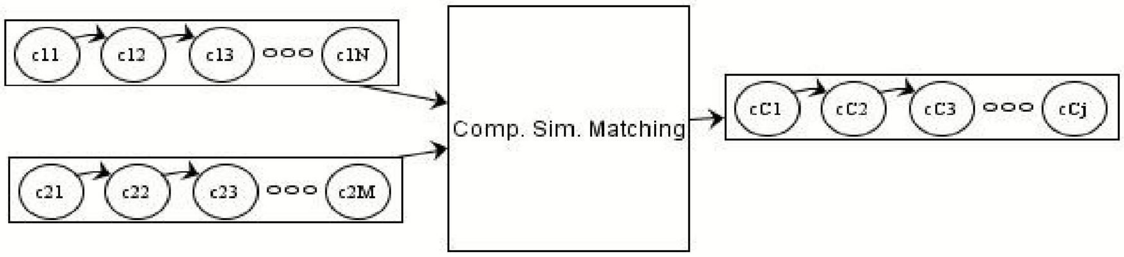

Figure 5. Component Sequence Similarity Matcher.

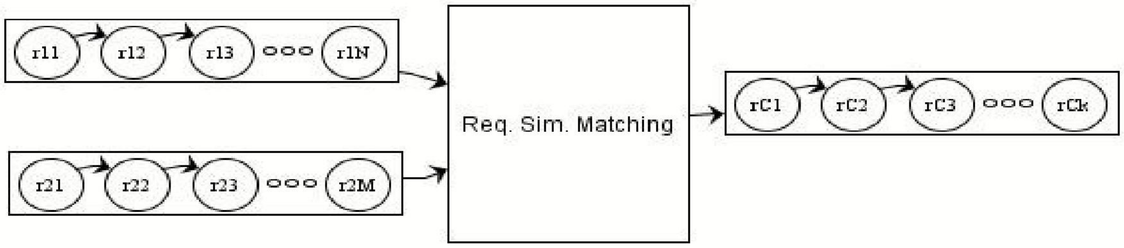

Figure 6. Requirement Sequence Similarity Matcher.

IV. ILLUSTRATIVE EXAMPLES

In this section example of presented approach is given. Assume that there are two bugs given and the proposed approach is applied on these bugs and similarity measurement is done.

Example 1: Clicking on close button of the application, a popup dialog should appear with the message "Do you want to quit?" with the option "Yes" and "No". Clicking on Yes, application will exit else not.

Applying the proposed approach - Transforming the bug into Event –Component-Requirement structure:

$e_{11} \rightarrow$ Event: Mouse Click

$c_{11} \rightarrow$ Component: Button

$r_{11} \rightarrow$ Requirement: a popup dialog should appear

$e_{12} \rightarrow$ Event: Mouse Click

$c_{12} \rightarrow$ Component: Button -Yes

$r_{12} \rightarrow$ Requirement: Application will quit.

$e_{13} \rightarrow$ Event: Mouse Click

$c_{13} \rightarrow$ Component: Button -No

$r_{13} \rightarrow$ Requirement: Application will not quit.

Example 2: Click on File menu, exit option. It should ask for the confirmation of exit from user.

Applying the proposed approach - Transforming the bug into Event –Component-Requirement structure:

$e_{21} \rightarrow$ Event: Mouse Click

$c_{21} \rightarrow$ Component: File Menu

$r_{21} \rightarrow$ Requirement: File Menu Options are displayed.

$e_{22} \rightarrow$ Event: Mouse Click

$c_{22} \rightarrow$ Component: File Menu Exit Option

$r_{22} \rightarrow$ Requirement: a popup dialog should appear with YES and NO Options

$e_{23} \rightarrow$ Component: Button -Yes

$c_{23} \rightarrow$ Requirement: Application will quit.

$r_{23} \rightarrow$ Event: Mouse Click

$e_{23} \rightarrow$ Event: Mouse Click

$c_{24} \rightarrow$ Component: Button –No

$e_{24} \rightarrow$ Event: Mouse Click

$r_{24} \rightarrow$ Requirement: Application will not quit.

Applying the ECR similarity model for example one and two, the longest subsequence matching is $\{e_{11}, c_{11}, r_{11}, e_{12}, c_{12}, r_{12}, e_{13}, c_{13}, r_{13}\}$ and $\{e_{22}, c_{22}, r_{22}, e_{23}, c_{23}, r_{23}, e_{24}, c_{24}, r_{24}\}$ found. This indicates that GUI bug example one and two are similar, the requirement is same but they come from two different GUI events. The fixation of bug one will be almost same as the fixation of two, which also indicates that the same code can be used for these two given GUI bugs for two different events.

Now consider three more GUI bug examples here for better understanding of approach.

Example 3: The password field is currently not allowing restriction on minimum number of characters. It should have criteria, that there should be at least 6 characters in the password field.

Applying the proposed approach - Transforming the bug into Event –Component-Requirement structure:

$e_{31} \rightarrow$ Event: Key Pressed

$c_{31} \rightarrow$ Component: Password Field

$r_{31} \rightarrow$ Requirement: If number of characters entered are less than 6 then display a error message popup.

$e_{32} \rightarrow$ Event: Mouse Click

$c_{32} \rightarrow$ Component: Button -Ok

$r_{32} \rightarrow$ Requirement: Password Field should be erased and cursor will point to this field.

Example 4: The re-enter your password field in login screen, should have criteria for entering at least 6 characters for password.

Applying the proposed approach - Transforming the bug into Event –Component-Requirement structure:

$e_{41} \rightarrow$ Event: Key Pressed

$c_{41} \rightarrow$ Component: Password Field

$r_{41} \rightarrow$ Requirement: If number of characters entered are less than 6 then display a error message popup.

$e_{42} \rightarrow$ Event: Mouse Click

$c_{42} \rightarrow$ Component: Button -Ok

$r_{42} \rightarrow$ Requirement: Password Field should be erased and cursor will point to this field.

Example 5: The zip code text field in contact details screen should have criteria that it should have minimum 10 characters.

Applying the proposed approach - Transforming the bug into Event –Component-Requirement structure:

$e_{51} \rightarrow$ Event: Key Pressed

$c_{51} \rightarrow$ Component: Text Field (ZipCode)

$r_{51} \rightarrow$ Requirement: If number of characters entered is less than 6 then display a error message popup.

$e_{52} \rightarrow$ Event: Mouse Click

$c_{52} \rightarrow$ Component: Button -Ok

$r_{52} \rightarrow$ Requirement: zip code Field should be erased and cursor will point to this field.

Applying the ECR similarity model for example three and four, the longest similar subsequence matching is $\{e_{31}, c_{31}, r_{31}, e_{32}, c_{32}, r_{32}\}$ and $\{e_{41}, c_{41}, r_{41}, e_{42}, c_{42}, r_{42}\}$ found. And Applying the ECR similarity model for example three and five, the longest subsequence matching is $\{e_{31}, r_{31}, e_{32}, c_{32}, r_{32}\}$ and $\{e_{51}, r_{51}, e_{52}, c_{52}, r_{52}\}$ found.

So from this similarity results it is visible that GUI bug examples three and four are similar, two similar bugs for two different components. And bug examples three and five are same requirements with varying number of characters restriction on the text field component.

V. CONCLUSION

In this paper an event-component-requirement structure similarity based methodology is presented to discover similar and duplicate GUI bugs in bug database. The proposed approach is explained with example. The future work related to this paper can be integrating the semantic similarity measures for requirement sequences similarity, with the proposed model to discover the more accurate results for similar and duplicate bugs.

REFERENCES

[1] Angelos Hiliautakis, Giannis Varelas, Epimendis Voutsakis, Euripides G. M. Petrakis, Evangelos Milios Technical University of Crete(TUC), Greece: Information Retrieval by Semantic Similarity, Int'l Journal on Semantic Web & Information Systems 2006.

[2] Aslam, J. Frost, M.: An Information theoretic Measure for Document Similarity, In Proceedings of the 26the Annual International ACM SINGIR Conference on Research and Development in Information Retrieval (ACM Press) 449-450,2003.

[3] Giannis Varelas, Epimenidis Voutsakis, Paraskevi Raftopoulou: Semantic Similarity Methods in WordNet and their Application to Information Retrieval on the Web, ACM, WIDM'05, November 5, 2005, Bremen, Germany.

[4] Hersh, W. Information Retrieval: A Health & Biomedical Perspective (Second Edition, Springer-Verlag), chap. 8, 2003.

[5] Jurafsky, D. and Martin, J. Speech and Language Processing (Prentice Hall), 2000.

[6] Nicholas Jalbert, Westley Weimer: Automated Duplicate Detection for Bug Tracking Systems, 2008.

[7] Papineni, K. Roukos, S. Ward, T. Zhu, W. (2002) BLEU: a Method for Automatic Evaluation of Machine Translation, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, 311-318, 2002.

[8] Salton, G. McGill, M. : Introduction to Modern Information Retrieval, McGraw-Hill Book Company, 1983.

[9] Sean and Hojun: Automated Detection of Duplicate Bug Reports with Semantic Concepts, IEEE COMPSAC, 2008.

[10] Shuji Morisaki, Akito Monden, Tomoko Matsumura, Haruaki Tamada and Ken-ichi Matsumoto: Defect Data Analysis Based on Extended Association Rule Mining, 2007.

[11] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik and Jiasu Sun: An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information, ACM, ICSE'08, May 10–18, 2008, Leipzig, Germany.