Collaborative Filtering Recommender System for Test Suit Prioritization in Web Applications

Maral Azizi
University of North Texas
Denton, Texas
maralazizi@my.unt.edu

Hyunsook Do
University of North Texas
Denton, Texas
hyunsook.do@unt.edu

ABSTRACT

.

Keywords: Test case prioritization, regression testing, recommender system, risk measurement, code quality.

1. INTRODUCTION

During application's life it may change several times, and one change can affect the entire system. To avoid the undesirable change or bugs, system testers need to test the overall functionality of the system before deploy the new release of the system. One of the most common way to evaluate the system quality in sequence of release is regression testing. In regression testing, testers check the software to ensure that new changes have not introduced new faults or it did not effect the other parts of system. Applying regression testing before deploying new release of the software, make the change process safer and more confident for developers.

As software grow, the size of test suite grow in a same manner, eventually testing the entire system can become expensive and may take 40% of whole project budget[]. Moreover, it is not feasible to test every single function of the system especially for large scale systems it may takes several days to execute test cases. Several techniques have been proposed to solve this issue such as, test reduction which is a technique that select subset of test cases that can address most of the issues, test selection also is a similar to test reduction but the main difference is this technique, select test cases based on their correlation with the changes in tow versions of the subject, and test case prioritization [].

Test case prioritization is a technique to catch the faults earlier, by executing the most important test cases first. So far, many prioritization strategies have been proposed by researchers [] and many techniques have been purposed such as : greedy techniques [], change impact analysis [], clustering approaches[], user session based techniques [], etc. but most of the investigated studies are based on two main features: code coverage and code complexity metrics. Meanwhile, there are many other features of the software that can be a factor of failure or can be a hint to find the failure causes. One interesting feature of web application is that recording users interaction data is a way easier compare to other types of applications. Different types of data from users interaction can be monitored, such as users sessions, cookies, telemetry data, users IP address etc. Storing and collection log files and users sessions nowadays are less costly and more accessible compare to past decades, due to lower price of the IT services and hardware. We can store information in database or cloud system rather than log files which was a traditional way to storing system back up. Having all these types of data and facilities, enables testers to identify most frequently used components which impact the system strongly.

Previous studies shown that change history has more significant role in terms of finding faults than code metrics []. Although, there are several research have been done about the efficiency of change metrics for defect prediction, such as [] but, researchers only focused on measuring the causes and effect of the changes in software. Moreover, users session and telemetry data have been used as a factor for research in software testing. For example, Elbaum et al. [**?**] performed a study on regression prioritization focusing prioritization at a product level by using telemetry from installations. Their work was extended in several domain such as, performance issue detection [**?**], bug detection [**?**], reliability testing of rule-based systems [**?**] and failure reproduction [**?**]. Sampath et al. [**?**] applied multiple techniques for prioritizing test cases by using user sessions as a factor for their techniques.

Lets conciser a typical test prioritization based on change history. In this scenario, testing is based on executing any component that has been changed in new release of software [1-2], while this could be a minor change that does not have a considerable impact on the entire system, such as renaming a function or can be a component that is used seldom by users, etc. In another scenario, we reorder test cases based on the component frequency, meanwhile, there might be no change in this particular component compare to previous version. For example, generally, core components in a system are mostly used components in the entire system while, those components due to their significant role have been tested several times and rarely change. Due to above challenges, we belive that current studies in test prioritization is neither efficient nor sufficient to address the test priritization problem.

In this paper we proposed an item based collaborative filtering recommender system for test case prioritization in web application which, uses the telemetry data as input for users rating and change history of applications as items information. The output of our recommender system is the prioritized test cases in a way to obtain the better results in terms of finding faults earlier. We applied our recommender system in three open source system and one commercial system. Our hypothesis is: Combination of various metrics can lead us to design more efficient model for fault detection. Our approach focuses on both users interaction and change history metrics to detect most risky components, then by automating the

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

detection of regression issues, we are reducing the analysis effort for test prioritization. Our recommender system, may not locate the regression issue sources directly, but it will provide a list of top potential risky components.

The main contributions of this study are as follows: 1) Proposing a new hybrid test prioritization technique for web applications domain, 2) Empirical evaluation of proposed technique and three other control techniques, and 3) Highlighting issues occurred during applying the proposed technique and its limitation and guidance to testers upon the results of our study.

The rest of the paper is organized as follows. In Section 2, we discuss the approach used in the research as well as formally define collaborative filtering recommender systems. In Section 3, we detail the empirical study that we performed. Section 4 provides the results of the study. Section 5 discusses the results and the implications of these results, and Section 7 presents related work. Finally, in Section 8, we provide conclusions and discuss future work.

2. PRIORITIZATION TECHNIQUES

2.1 Recommender System Approach

In this section, we explain our proposed technique, algorithm and the workflow of our recommender system.

2.1.1 An Overview of Our Proposed Technique

The main focus of collaborative filtering recommender system is the users rating on existing items. In our study we inspired by the concept of this idea to identify the most important components of our subjects. Our proposed technique is based on two main ideas: first, the most used components of the system have more significant role in the system overall quality and second, change history is a key role of detecting the cause of recent faults of the system. Figure 1 shows the process of recommender system. Matrix in the picture returns how many times each component have been used by an specific user. To generate the list of Top-N components we need to calculate the similarity between users usage patterns and components.

The goal of our hybrid recommender method is to suggest the highest risky components with most frequency among all other components in the applications. To do this we used two collected datasets: user sessions $U = \{u_1, u_2, ..., u_m\}$ and list of our components $C = \{c_1, c_2, ..., c_n\}$. Each user u_i has a list of components c_{ui} which the user has performed at least one task on that particular c.

Generally in recommender systems prediction is based on the numerical value of rating from an active user, while in our case we do not have a access to such rating module instead we assume the value of frequency access for a specific component by an individual active user as a rating score.

There are two main techniques for collaborative filtering algorithms: Memory based (user-based) and Model based (item-based) algorithms. In memory based collaboration filtering we seek to find the most similar users to the current active user while in the model based algorithm first we design a model of users rating and then we will evaluate the expected rating of an item based on the previous rating of the other similar items.

The item based approach looks into the set of items the target user has rated and computes how similar they are to the target item i and then selects k most similar items $\{i_1, i_2, ..., i_k\}$. At the same time their corresponding similarities $\{s_{i1}, s_{i2}, ..., s_{ik}\}$ are also computed. Once the most similar items are found, the prediction is then computed by taking a weighted average of the target users's rating on these similar items. In order to determine similarity between two components i and j we used Pearson-r correlation. If U is the set of users who rated i and j then we compute the correlation similarity by

$$Sim(i,j) = \frac{\sum_{u \in U}(C_{u,i} - \bar{C_i})(C_{u,j} - \bar{C_j})}{\sqrt{\sum_{u \in U}(C_{u,i} - \bar{C_i})^2}\sqrt{\sum_{u \in U}(C_{u,j} - \bar{C_j})^2}}$$

where $C_{u,i}$ is the value of frequency access of user u on the component i, $\bar{C_i}$ is the average value of frequency for the $i-th$ component .

$$P_{u,i} = \frac{\sum_{all\ simillar\ components,N}(S_{i,N} * R_{u,N})}{\sum_{all\ simillar\ components,N}(S_{i,N})}$$

In order to estimate the rate of ignored components we will conduct ratio prediction computation by using weighted sum method. This method provides the ratio prediction of a specific component i for user u based on similar components to i. this method will return how a user rates the similar items.

2.1.2 Workflow of Recommender system

In figure 2 we depict the workflow of our proposed technique. The workflow steps are as follows:

1) Users sessions collection: This module collect the telemetry data from users interaction. The obtained information are:user name, date, action name and page name.

2) Change history collection: This module obtains change history for each component. In section xx we describe the metrics and process of change history collection.

3) Determine the most frequent components: This module calculate the most access component based on users interaction. To compute the frequency score we applied to different techniques which have been explained in section xx.

4) Components similarity computation: In order to measure similarity between two component we used Pearson-r correlation formula which we described above.

5) Ranking components:Normally in CF recommender system there is a prediction step which calculate the probability of most similar item for an active user since, the focus of our study is identifying the most risky component we eliminate that step. Using the below formula we can calculate the ranking score for each component:

$$R_{Ci} = F_{Ci} * W_{Ci}$$

where F_{Ci} is the frequency score of component i and W_{Ci} is the weight of C_i.

6) Test case recommendation: The final step of our recommender system is reordering our test cases based on their coverage of $Top-N$ components.

2.2 Control Techniques

3. EMPIRICAL STUDY

3.1 Objects of Analysis

To investigate our research questions, we performed an empirical study. The following subsections present our objects of analysis, study setup, threats to validity, and data analysis.

3.2 Variables and Measures

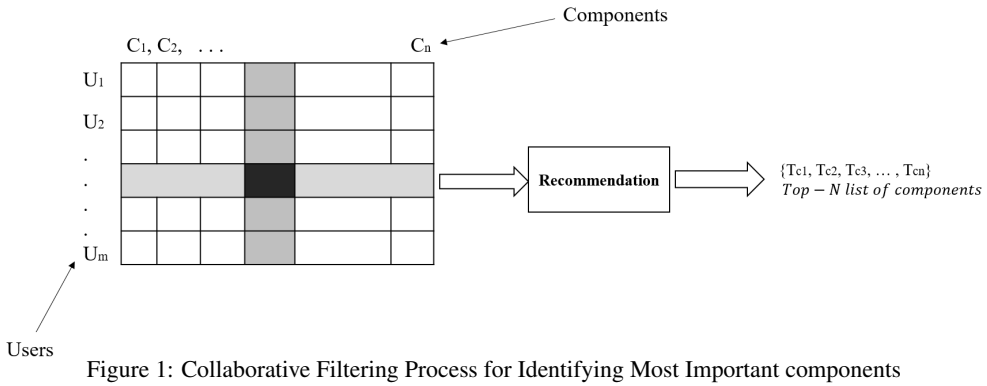

Figure 1: Collaborative Filtering Process for Identifying Most Important components

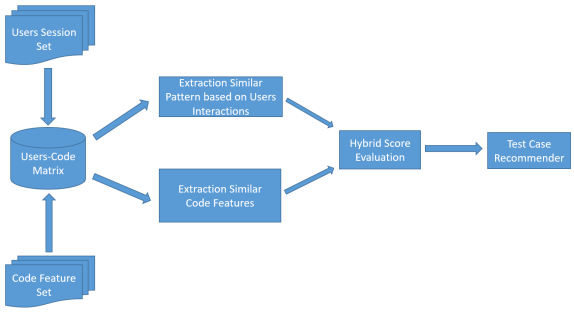

Figure 2: The workflow of the proposed risk evaluation method

Table 1: Subject Applications properties

Metrics	DASCP	MixERP	nopCommerce	Coevery
Classes	107	5,760	1,919	2,258
Files	201	5,473	1,473	1,875
Functions	940	57,594	21,057	13,041
LOC	35,122	837,674	226,354	120,897
Sessions	748	xx	xx	xx
Faults	35	xx	xx	xx
Version	3	5	23	3
Test Cases	95	21	543	1,120
Installations	3	1	2	1

3.2.1 *Independent and Dependant Variables*

Independent variables in our study are the user-session-based test suites, the seeded faults and the test case prioritization techniques. Dependent variables are rate of fault detection, average percent of faults detected (APFD) [xx], and test execution time.

4. EXPERIMENT SETUP

In this section, we describe applied techniques to examine different types of test prioritization techniques. Our Study contains three main categories:

User Session based Prioritization In this experiment we reorder our test cases that cover most frequently used web pages. Assume (T, S, M, f, A), where T is the set of test cases, S is the collection of user session, M is applied method, f is the frequency score based on M and A is Average percentage of fault detection, the desired output is to reorder T based on f where A obtains higher value compare to its default value which is a random.

Change History based Prioriitization We used the change history of our AUTs to determine most risky components then we reorder the test cases based on the ascending value of risk for each component.

Hybrid Recommendation Method In our third study we proposed a hybrid method which use the users interaction data and change history. We draw similarity matrix based on those information to find the most risky and frequently used web page in our web application. Afterward, we change the order of our test cases based on output of our recommender system.

4.1 User Session based Prioritization

Our first test prioritization is based on the information we collected from monitoring our volunteer users. We asked hundred volunteer users whom are computer science students from University of North Texas and we asked the to perfome variety of tasks in our applications. Among all collected information, we applied two main techniques to figure out most frequent parts of our applications. Below we describe these two techniques:

4.1.1 *Most Frequent Web Pages*

this approach determines the most frequent web pages that have been used by users. We assume that the most frequent web pages are playing more important role in the system, eventually, they should have a higher priority to be test first. Another reason of why frequency of web pages is the key for testing is most frequent web pages usually contains more functionality and connections with other pages, so any bug in those pages can effect higher portion of the entire system. Here we define base session as a session that provided by our users and covers the most interactions in the application. we picked 20 percent of our total sessions as a base session. For example, in online shopping, our base session is a sequence of actions from user login to check out, including all necessary actions and some random unnecessary such as browsing other items, checking inbox during shopping process etc.

After collecting all sessions, we condoct an analysis of web pages frequency by comparing obseverd web pages in each session with base session.

$$F_{w,i} = \frac{\sum_{page\ score\ for\ each\ session}(S_i)}{\sum_{number\ of\ base\ sessions}(BS_i)}$$

Using figure 3 as an example, page x from base session have been observed in both test session 1 and 2. If we assume we have 10 test session and 3 base sessions and this particular page have been observed in 7 of the 10 pages then, frequency of page x is equal to $(0.7/3) = 0.23$

Figure 3: Sample of Base Session and Test Sessions

Table 2: Change metrics used to evaluate risk in this study

Metrics name	Description
Revision	Number of revision of a component
LOC Added	Added lines of code
Max LOC Added	Maximum added lines of code
AVE LOC Added	Average added lines of code
LOC Deleted	Deleted lines of code
Max LOC Deleted	Maximum deleted line
AVE LOC Deleted	Average deleted lines of code
Code Churn	sum of change in all revisions
Age	Age of the component in days from last release
Time	Time of the change in dd-mm-yyyy format

4.1.2 Most Frequent Actions

Most frequent actions typically are the main functions in the system and in a normal case they contain high dependency to the other classes and functions. If one main function fails it can be cause of a significant failure or degrade of the system. Another factor of importance of most frequent functions is to prevent of domino effect in the system.

4.2 Change History based Prioriitization

Among hundreds attribute of code and change history to evaluate the code quality and error proness we choose change history to identify most risky web pages. In previous study [ref Raimund] shows the role of change history in bug detection and its superiority in terms of bug detection. In this study we collect the change history of our three applications. We chose xx metrics based on their correlation to the previous observed bugs. Also most of these metrics have been used in [,,,,] as a feature for bug detection. Table xx shows the applied change metrics in this study.

After the change history data collection, we applied ANOVA analysis on our dataset to attain a linear model. Our aim is to find the correlation coefficient for each metric to measure statistical relationships between a variable and real defects. The value of this measure fluctuate between 1 and -1. Here 1 indicates strong positive relationship which means increase in variable values, increases the output value, 0 shows no correlation and negative value means revers correlation. Other than that, we calculated root mean squared error and mean absolute error. The lower error value shows that the model has higher prediction accuracy. Our label in the dataset is the bug reports. Note that our web pages are dynamic and they contain a class code behind which is the main concern in our study not HTML bugs.

In order to evaluate our linear model, we applied 10-fold cross validation. In next step we applied our obtained model to evaluate the riskiness of each component.

4.3 Hybrid Recommendation Method

5. STUDY

5.1 Object of Analysis

5.2 Variables and Measures

5.3 Data Collection and Experimental Setup

6. THREATS TO VALIDITY

7. DATA AND ANALYSIS

Table 3: Test Case Prioritization Techniques

Group	Technique	Description
Control	Change History	Test Case Prioritization based on Cl
	Most frequent web pages	Test Execution based on value of fr
	Most frequent actions	Test Execution based on value of fr
	Random	Test Execution in Random Order.
Heuristic	Hybrid Collaborative filtering	Test Case Prioritization based on Pr

8. DISCUSSION

9. RELATED WORK

9.1 Web applications

9.2 Test Suite Prioritization

9.3 Risk Measurement

10. CONCLUSIONS AND FUTURE WORK

Acknowledgments

11. REFERENCES