Predicting Risky Program Source Files

Syed Nadeem Ahsan, Syed Haider Abbas Naqvi and Kamran Raza
Faculty of Engineering Sciences and Technology
Iqra University, Karahi, Pakistan
sn_ahsan@yahoo.com, {haider0202, kraza}@iqra.edu.pk

***Abstract*—Change in source codes is an essential and routine activity of software development and maintenance. It has been observed that this activity might result in faults that might harm the use of the software. Therefore, it is always useful for software managers and programmers that before making any changes in source files, they should know the degree of risk associated with changing source files. In this paper, we present our approach to identify source files, which are risky or at least sensitive to new changes. We defined a set of metrics to compute the degree of risk associated to a source file. To validate our approach, an experiment has been performed by using Mozilla project's data. The experimental results show that the source files having higher risk values are more risky when applying the next change and thus should be tested more thoroughly.**

***Keywords*-Software evolution; code metrics; risk estimation;**

I. Introduction

In the field of software engineering, research studies reveal that over 90% of the software development cost spends on software maintenance and evolution. Moreover, software testing consume most of the development time and money [6]. Therefore, in today's software industry, software quality assurance personnel need those techniques which predict risky source code modules, so that more thorough testing can be performed for risky source code modules. Moreover, a developer can use such predictions to focus quality assurance activities.

In recent years, much research work has been performed to build models to predict risky or faulty source code modules. Mostly, researchers extract knowledge from the repository of software evolution [3][14]. This knowledge have been used to build models to predict faulty source code modules in the new release of a software product [2][7][9]. Therefore, the goal of our research work is to use the software evolution data, and define a new set of metrics to compute the degree of risk associated to the new changes in the source files. Such information is not only useful for developers but also for software managers in order to assign resources, e.g., for testing.

It has been found that the history of software change patterns might be used for the analysis of risky or faulty program files [1][10]. Therefore, in our research work, we first classify the changes in the source code into four well known types of software changes i.e., clean-changes, bug-introducing changes, bug-fix and introducing changes, and finally bug-fix changes [7]. Then, we define a set of metrics using the four different types of software changes. Finally, we derive an empirical relation for the risk model. To validate our research approach, we performed an experiment by extracting the change history data of source files from the software repository of Mozilla Project. After extracting the data, we applied an approach that allows us to identify the different types of code changes like, bug fixing, clean, bug introducing, and bug fix-introducing transactions [7]. We used the data of software changes and computed a set metrics. Finally, we used these metrics in our propose risk estimation model and obtained the degree of risk associated to source files.

The paper is organized as follow: In Section 2, we discuss related work. In Section 3, we describe the types of software changes. In Section 4, we describe our approach. In the next sections, i.e., Section 5 and 6, we discuss the obtained empirical results and conclude the paper.

II. Related Work

Software evolution data have been used to construct models for the classification of clean and faulty source code [3][14]. Asundi [8] highlight the importance and the major challenges of the risk estimation model of program source files. Whereas, most of the research works are based on data driven techniques and used machine learning algorithm to build model for the prediction of faulty source code modules [13]. It has been observed that very few attempts have been made to build empirical models for the computation of risk factor associated with the new changes in source codes [1][2].

Porter and Selby [4] used classification trees based on metrics from previous releases to identify components having high-risk properties. Mockus and Weiss [2] presented a model to predict the risk of new changes, based on previous information. The authors modeled the probability of causing failure of a change made to software, using properties of a change as model parameters. Sliwerski et al. [7] analyzed the CVS repository of Mozilla and Eclipse together with the information stored in the corresponding Bugzilla bug reporting system to identify fix inducing changes. Robert et al. [10] used code churn metrics (metrics which are based on addition, subtraction and modification of source code lines) for predicting faults in software. They found that change in the prior release is an essential component of fault prediction method. Similarly, our research work is focused to use the source code modification data, but instead of using direct code churn metrics, we used a new set of metrics, and derived a relation for the risk model of program source file. The research work present in this paper is the further enhancement in our previous risk model [12].

III. Types of Changes in Source Code

In this paper, we propose an approach to build model for the prediction of risky source code module. Our approach is based on software repository data, i.e., version controlling system (CVS) and bug tracking system (Bugzilla) [14]. First, we process the software repository data, and compute a set metrics using the following four types of source code changes: clean, bug introducing, bug fixing and bug fix-introducing changes. To obtain these types of source code changes, we used an approach which was given by Sliwerski et al. [7]. After obtaining the metrics data, we use these metrics in our risk model, and compute the risk factor associated to each source files. In the following paragraph we briefly describe the four different types of changes in source code (for details see [3][7]).

1) **Bug Fixing Change:** This type of changes are occurred when developers want to change the source code to fix a bug.
2) **Bug Introducing Change:** Once we identify that the change is a bug fixing change in the source file, then it is required to locate those changes in the source file revisions, which actually introduced the bug.
3) **Bug Fix-Introducing Change:** After identifying all the Bug-Introducing and the Bug-Fixing changes, then the next step is to list all those source file changes, which have both type of changes, i.e., Bug Fixing and Bug Introducing Changes.
4) **Clean Change:** Finally, the set of source file changes, which are not identified as bug fix or bug introducing or bug fix-introducing, are listed as clean changes.

IV. Our approach to Build a Risk Model

Our approach is based on a mathematical model, which can be derived using the definition of risk. In engineering risk corresponds to the costs in case of an accident [5], and according to IEEE, the standard definition of risk is: "An expression of the impact and possibility of a mishap in terms of potential mishap severity and probability of occurrence (MIL-STD-882-D, IEEE 1483)." Whereas, in our approach, "accident" or probability of "mishap" is basically the probability of faulty change in source code, which may introduce one or more bugs in a software, and "costs" is the time or effort needed to fix those bugs. Therefore, we process the data of source code's change history, and compute the probability of faulty change of each source file. Finally, to obtain the risk value, we multiply the probability of faulty/buggy changes with the costs factor, and compute the risk value. Hence, in our case, risk can be defined as:

$$\text{Risk} = (Prob.\ of\ Bug) \times (Bug\ Impacts/Costs) \quad (1)$$

$$\text{Risk}_j = \text{p}_{BUG} \times \text{C}_j \quad (2)$$

where, Risk_j is the risk that a change in a source file could be a faulty change, and it could introduce j number of bugs. p_{BUG} is the probability that a new change could be a faulty change, and C_j represent the expected impact/costs of that faulty change. C_j represent the impact in term of an estimated costs that need in fixing the introducing number of faults. A faulty change in a program file may introduce *j* number of faults into a software system. C_j is the costs of fixing *j* faults. Therefore, the risk that a faulty change may introduce *j* faults is given by equation (2).

The probability of a change to lead to a bug (or accident), i.e., p_{BUG} can be obtained from the history of *bug introducing* and *bug fix-introducing* changes of a source file. For this purpose, the number of bugs introducing (n_I) and bug fix-introducing changes (n_{FI}) has to be divided by the number of all changes (n_C), e.g.,

$$\text{p}_{BUG} = \frac{\text{n}_I + \text{n}_{FI}}{\text{n}_C} \quad (3)$$

Now, to find the risk value of a source file, we have to know the expected costs of the bugs that could be generated. Let, p_j be the probability of introducing *j* bugs by a faulty change, and c_j is the costs of fixing *j* bugs.

$$\text{C}_j = \text{p}_j \times \text{c}_j = \frac{\text{n}_j}{\text{n}_C} \times \text{c}_j \quad (4)$$

$$\text{Risk}_j = \frac{\text{n}_I + \text{n}_{FI}}{\text{n}_C} \times \frac{\text{n}_j}{\text{n}_C} \times \text{c}_j \quad (5)$$

where, n_j is the number of changes that introduced *j* bugs per change. Its value can be obtained from the previous history of source file. If we consider the costs of each bug as a constant value, i.e., *c*, then Eq. (5) can be written as,

$$\text{Risk}_j = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{\text{n}_j}{\text{n}_C} \times \text{j} \times \text{c} \quad (6)$$

In case of Open Source Software (OSS), there is no information regarding the costs for correcting a fault in terms of amount of work necessary, and moreover there is no information available about costs related to the bug when the program is executed. Hence, we might assume unit costs for each fault, i.e., *c*=1.

$$\text{Risk}_j = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{\text{n}_j}{\text{n}_C} \times \text{j} \quad (7)$$

Equation (7) can be used to find the risk associated with source files, which could generate *j* faults. Every change might lead to several bugs. Hence, the expected costs can be estimated by averaging the costs of correcting "k" faults multiplied with the probability that a change causes "k" faults, i.e.,:

$$\text{Risk}_{avg(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{1}{\text{k}} \sum_{j=1}^{k} \frac{\text{n}_j}{\text{n}_C} \times j \quad (8)$$

$$\text{Risk}_{avg(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{1}{\text{k}} \times \frac{1}{\text{n}_C} \times \sum_{j=1}^{k} (\text{n}_j \times j) \quad (9)$$

where, $\sum_{j=1}^{k}(n_j \times j) = n_B$, and n_B is the total number of bugs occurred during the life of a source file. Its value can be obtained by adding the value of bug counts n_{BI}, which is generated by bug introducing changes and bug count value n_{BFI}, which is generated by bug fix-introducing changes (n_B=n_{BI}+n_{BFI}).

$$\text{Risk}_{avg(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{1}{\text{k}} \times \frac{\text{n}_B}{\text{n}_C} \quad (10)$$

where, "k" is the maximum number of faults introduced by a single change in a source file. Furthermore, equation (10) can be used to compute the maximum risk instead of an average risk.

$$\text{Risk}_{max(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{\text{n}_B}{\text{n}_C} \quad (11)$$

Equation (11) may be used to measure the maximum risk associated with the new changes in a source files. In equation (11), $\frac{n_B}{n_C}$ is the average number of faults per change. It may be written as $\text{C}_{max(j)}$. In our experiment, we used equation (11) to measure the risk value. In equation (11), we considered unit cost i.e., *c*=1, if we do not consider unit cost then equation (11) may be written as,

$$\text{Risk}_{max(j)} = \frac{\text{n}_I\text{+ n}_{FI}}{\text{n}_C} \times \frac{\text{n}_B}{\text{n}_C} \times \text{c} \quad (12)$$

Equation (12) is the main equation to compute the maximum risk factor associated with source file. The major challenge is to compute the cost value, i.e., "c" in equation (12). Since, in case of open source, the costs related to software change is not recoded, therefore, it is difficult to compute the costs value of software changes. However, we can make some assumption for costs value, like we did in equation (7). Moreover, it has been found in literatures that some attempts have been made to compute the effort or costs value of software changes [8] [11]. We are still working to find some better technique to compute the costs value of software changes. Now, In the next section, we discuss some experimental results, which are based on Eq. (1) to Eq. (11).

V. Results and Discussion

In order to validate our approach, we performed an experiment by downloading source files and bug reports data from the Mozilla project (evolution data is freely available on website). After downloading, we used an approach to process the downloaded data, and find the distribution of four different types of changes in source codes (for details see [7] [12]). Our experimental results are shown in Table I.

Table I depicts the obtained results of risk associated with 7 different source files of Mozilla project. For example, consider a source file, i.e., CBrowserShell.cpp from Table I. The number of fault-introducing changes and the number of fault-fix-introducing changes are 17 and 49 respectively. Hence, the probability of a change to introduce at least one bug in this example is: p_{BUG} = (n_I + n_{FI}) / n_C = (17+49)/93= 0.71. Moreover, the data of Table I, can easily be used to compute the expected cost of fault fixing in source file. Like in case of CBrowserShell.cpp source file, the cost is $\text{C}_{max(j)} = \frac{n_B}{n_C} = \frac{69}{93}$ = 0.74. From this, the risk factor associated with the new changes in the source file CBrowserShell.cpp can be computed as follow: $\text{R}_{max(j)}$ = p_{BUG} × C = 0.71 × 0.74 = 0.53.

Similarly, the risk can be computed for each source files. The results of this computation on sample data set of seven source files are given in Table I. Figure 1 shows the normalized risk factor associated with each source files. In our sample data set, the source file calDateTime.cpp is

Table I
COMPUTATION OF RISKY SOURCE FILES USING THE DISTRIBUTION OF FOUR DIFFERENT TYPES OF SOURCE FILE CHANGES

Mozilla Source Files (.cpp)	Four types of changes				Total Change	Total Faults	Total Faulty Changes	p_{BUG} = (n_I+n_{FI}) / n_C	$C_{max(j)}$ = n_B / n_C	$Risk_{max(j)}$ = $p_{BUG} \times C_{max(j)}$
(sample data)	Clean	Fix	n_I	n_{FI}	n_C	n_B	n_I+n_{FI}			
calDateTime	2	5	21	82	110	112	103	0.94	1.02	0.95
calICSService	1	6	9	60	76	73	69	0.91	0.96	0.87
CBrowserShell	11	16	17	49	93	69	66	0.71	0.74	0.53
CNavDTD	42	46	147	263	498	418	410	0.82	0.84	0.69
COtherDTD	8	20	57	65	150	128	122	0.81	0.85	0.69
dlldeps	32	44	35	92	203	132	127	0.63	0.65	0.41
EmbedPrivate	15	33	20	84	152	110	104	0.68	0.72	0.50

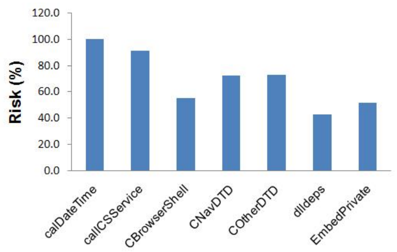

Figure 1. Risk factor associated with seven source files of Mozilla project

the most risky source file for further changes. Whereas, the source file dlldeps.cpp is the less risky file for further changes. A graph which is similar to Figure 1 can be used during software maintenance phase to identify those source files which have a high risk factor. Such information helpful to perform software maintenance task.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented our approach to build a mathematical model, which computes the risk factor associated with the new changes in source codes. Our risk model is based on a set of metrics. We obtained these set of metrics by using the evolution data of source codes. Furthermore, to validate our risk model, we performed an experiment by using the software evolution data of Mozilla project. We found that our model successfully measured the risk factor associated with source files. The source files which have higher risk factor values are more risky when applying the next change and thus should be tested more thoroughly. Currently, we are working to develop a method which can be used to compute the actual costs/impact value of a bug. Moreover, in future, we will further enhance our risk model by adding more metrics.

REFERENCES

[1] A. A. Phadke and E. B. Allen, “Predicting Risky Modules in Open-Source Software for High-Performance Computing,” In proceeding of IWSEHPCS, 2005.

[2] A. Mockus and D. M. Weiss, “Predicting risk of software changes,” Bell Labs Tech., vol. 5, Apr-June 2000, pp 169-180.

[3] M. Fischer, M. Pinzger, and H. Gall, “Populating a release history database from version control and bug tracking systems.” In Proceeding of the International Conference on Software Maintenance (ICSM), 2003, pp 13-23.

[4] A. A. Porter and R. W. Selby, “Empirically Guided Software Development Using Metric-Based Classification Trees,” IEEE Software, volume 7(2), 1990, pp 46-54.

[5] E. Addison, “Managing Risk: Methods for Software Systems Development(Sei Series in Software Engineering),” Addison Wesley, February 28, 2005.

[6] L. Erlikh, “Leveraging legacy system dollars for e-business,” IT professional, volume 2(3), 2000, pp 17-23.

[7] J. Sliwerski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?,” Proceedings of the 2005 international workshop on Mining software repositories, 2005, pp 1-5.

[8] J. Asundi, “The need for effort estimation models for open source software projects.” In Proceedings of the fifth workshop on Open source software engineering (5-WOSSE), ACM, New York, NY, USA, 2005, pp 1-3.

[9] K. A. Gunes and L. Hongfang, “Building Defect Prediction Models in Practice,” IEEE Software, 22(6), 2005, pp 23-29.

[10] R. M. Bell, T. J. Ostrand, and E. J. Weyuker, “Does measuring code change improve fault prediction?,” In Proceedings of the 7th International Conference on Predictive Models in Software Engineering (Promise), 2011.

[11] S. N. Ahsan, J. Ferzund and F. Wotawa, “Program File Bug Fix Effort Estimation Using Machine Learning Methods for OSS,” 21st Int. Conference on Software Engg. and Knowledge Engg. (SEKE), Boston, USA, July 1-3, 2009, pp 129-134.

[12] S. N. Ahsan, J. Ferzund, and F. Wotawa, “A Database for the Analysis of Program Change Patterns,” In proceeding of the 4th International conference on Networked Computing and Advanced Information Management, 2008.

[13] S. Kim, E. J. Whitehead, and Y. Zhang, “Classifying Software Changes: Clean or Buggy?,” IEEE Transactions on Software Engineering, March/April, 2008, pp. 181-196.

[14] T. Zimmermann, P. Weisgerber, S. Diehl, A. Zeller, “Mining Version Histories to Guide Software Changes,” In Proceedings of the 26th International Conference on Software Engineering, May 23-28, 2004, pp 563-572.