A Dynamic Approach to Software Bug Estimation

Chuanlei Zhang†, Hemant Joshi†, Srin Ramaswamy*, Coskun Bayrak†
†Applied Science Department, †Department of Computer Science
University of Arkansas at Little Rock
Little Rock, AR 72204
{cxzhang, hmjoshi, srini, cxbayrak}@ualr.edu

Abstract—There is an increase in the number of software projects developed in a globally distributed environment. As a consequence of such dispersion, it becomes much harder for project managers to make resource estimations with limited information. In order to assist in the process of project resource estimations as well as support management planning for such distributed software projects, we present a methodology using software bug history data to model and predict future bug occurrences. The algorithm works in a two-step analysis mode: Local and Global Analysis. Local analysis leverages the past history of bug counts for a specific month. On the other hand, global analysis considers the bug counts over time for each individual component. The bug prediction achieved by the algorithm is close to actual bug counts for individual components of Eclipse software.

I. INTRODUCTION

Traditionally, most software projects employ centralized software development processes, normally performed at a single location. All of the developers usually work in the same city, or even in the same building. Project management and resource allocation are made based on well-established product plans. This form of a centralized software development process and project management has been successfully used for decades. The globalization of the software industry is has led to products being developed globally, where the software development teams and the project management teams work in disperse geographical locations [1]. Despite the fact that global software development can bring benefits such as improving the product quality by hiring top rated international IT professionals and lowering the development costs by using services from countries with cheaper cost of labor / living, it creates several software engineering challenges due to the impact of temporal, geographical and cultural differences among distributed development and management teams [2] [3].

One of the challenges in global software development is project management and appropriate resource allocation: (1) since the management team and the development team might work at different locations, problem identification, issue resolution, and decision making are not as straightforward as that of the centralized software development scenario; (2) The tasks assigned to distributed teams must be clear and specific, because distributed teams are not as flexible as centralized teams. The hiring of the distributed developers is based on their expected expertise and potentials to rightly fit the foreseen tasks in order to minimize development costs. Therefore, in global software development, it is critical to accurately identify the task and estimate the cost of software development at different sites and allocate appropriate resources accordingly [4].

Software repositories, such as source control systems, defect tracking systems, and communication archives, contain rich records of the software development history. Such data is available for most software projects and can be used to extract valuable information to guide software development and project management. For example, archived e-mail can be used to study the communication network of the project [5] [6], change logs can be used to study the evolution of the source code, and so on [7].

One type of valuable information that can be extracted from repositories is change propagation, which is to determine the group of components should be changed together because of the interdependencies between software components [8]. For example, Zimmermann et al. [9] applied data mining to software version history and implemented a tool to detect related changes in open-source software including Eclipse€, GCC, and Python. Ying et al. [10] applied association mining techniques on Eclipse and Mozilla. Their association rules can be used to predict future co-changes of software components. The prediction of software change propagations can assist the developers to correctly maintain a software product and the managers to accurately estimate the amount of work involved in a maintenance task and efficiently allocate the necessary resources.

Another type of valuable information that can be extracted from software repositories is bug patterns. Bugs are inevitable consequence of a software product development process. The correct prediction of the number of bugs and the type of bugs in a software product can greatly assist in project estimation and management [11] [12]. In this paper, we describe how bug history logs can be used to model the occurrence of bugs and how this pattern can assist the project management to plan for and estimate resource needs in a globally distributed development environment.

The rest of the paper is organized as follows. Section 2 reviews related work on mining bug repositories. Section 3 entails the underlying algorithm for bug prediction. In section 4, we discuss the results of the prediction. Finally, section 5 presents our conclusions and avenues for future research work.

† http://www.eclipse.org
II. RELEVANT WORK

To remain competitive in the fast growing and changing world of software development, project managers must optimize the use of their limited resources to maximally deliver quality products on time and within budget [13]. Software bug repositories have been recognized as data assets that can provide useful information to assist with software development and project management. Considerable research has been performed in this area.

One line research is to characterize the information stored in bug repositories. Anvik et al. [14] analyzed the information stored in a bug repository and discussed how to use it. Sandusky et al. [15] identified bug report networks, which are groupings of bug reports due to duplication, dependency, or reference relations. Mockus et al. [16] used bug information to determine the various roles people played in the Apache and Mozilla projects. Crowston and Howison [17] used bug repository information to analyze the social structure of open source projects.

Another line of research is to model the bug patterns. Askari and Holt [18] analyzed the data from open source repositories, including defect tracking system, and developed three probabilistic models to predict which files will have bugs. These include: (i) Maximum Likelihood Estimation (MLE), which simply counts the number of events, i.e., changes or bugs, that happen to each file and normalizes the counts to compute a probability distribution. (ii) Reflexive Exponential Decay (RED), which assumes that the predictive rate of modification in a file is incremented by any modification to that file and decays exponentially. (iii) The third model, the RED-Co-Change model, with each modification to a given file, not only increments its predictive rate, but also increments the rate for other files that are related to the given file through previous co-changes. The performance of the different prediction models were evaluated using an information-theoretic approach. Ostrand and Weyuker [19] [20] [21] [22] developed a statistical model that uses historical fault information and file characteristics to predict which files of a system are likely to contain the largest numbers of faults. This information can be used to prioritize the testing components to make the testing process more efficient and the resulting software more dependable.

Still another line of research is to predict the number of bugs and types of bugs in a software product. Gravel et al. [23] used project change history to predict fault incidences. Menzies el al. [24] performed a case study on NASA’s Metrics Data. Through mining the bug history, they built defect detectors to predict the potential number bugs for different software components. Their findings can help project planning and resource allocation. Williams and Hollingsworth [25] presented a method to create bug-finding tools that is driven by the analysis of the previous bugs. They showed that past bug history of a software project can be used as a guide in determining what types of bugs should be expected in the future. Moreover, their approach can also help identify which groups of bug reports are more reliable.

III. BUG PREDICTION

The lifetime of a bug in a post-delivery product can be divided into two stages, hidden stage and visible stage. The hidden stage starts as soon as a product is released. (Although bugs might enter a software product in different phases of the development process, in this study, only bugs in the released product are considered.) When a bug is identified and reported, it enters into the visible stage. When the bug is killed (the fault is corrected), the visible stage ends [26]. One parameter to measure the bug life is the bug hidden time, which is the duration that a bug is in the hidden stage. Bug hidden time is affected by many factors. For example, a bug in a widely used product may be quickly detected and reported by users, and accordingly has a shorter hidden time. Clearly, the number of bugs in a released product is fixed. Different bugs might have different hidden time, and accordingly, the number of bugs detected and reported each month after the product release is affected by many factors, such as the bug’s concealment, the extent usage of the product, and the user skill level. However, most of these affecting elements are not measurable. Therefore, there is a need of an empirical approach that can predict bugs based on bug finding history instead of based on these not measurable factors.

In this paper, we present a bug prediction algorithm. The algorithm is intended to be used to predict the number of bugs to be detected and reported every month. First we make the hypothesis that bug prediction of any given component for a given month, i is most impacted by the bug-count reported for the previous month, (i-1). This can be represented as:

\[ \text{BugCount}_i = \alpha_i \times \text{BugCount}_{i-1} \]

So the bug prediction for any month mainly relies on the bug-count of previous month. We introduce the correction factor alpha (\(\alpha\)), which is a measure of how much the predicted bug-count deviates from the previous month. Hence,

\[ \text{BugCount}_i = \alpha_i + \text{BugCount}_{i-1} \]  

The correction factor \(\alpha\) is used as an offset adjustment to predict how much the bug-count will vary in the month under consideration from its predecessor. In order to decide the correction factor \(\alpha\), we take into account the bug-count history of all available years and also use local corrections. Thus, we study bug prediction problem by analyzing the historical bug data both locally, and globally.

IV. CASE STUDY

The case study was performed on Eclipse bugs repositories [27]. We downloaded over 70,000 bugs related to Eclipse project over past 6 years. We particularly were interested in 32 components listed in table 1.

<table>
<thead>
<tr>
<th>Equinox.Bundles</th>
<th>Platform.Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equinox.Incubator</td>
<td>Platform.IDE</td>
</tr>
</tbody>
</table>