A Model for Software Defect Discovery

Thesis Proposal

James Tunnell

November 1, 2014

1. Introduction

Search-Based Software Engineering (SBSE) is an emerging area of research, where search-based optimization techniques are applied to the Software Engineering problem domain. SBSE problems include The Next Release Problem, Structural Test Data Generation, and Software Cost Estimation[4].

To make SBSE of practical use on a software project, a method is needed to model the practical details of a project in an abstract way so that is amenable to search-based optimization. In this paper, such a model is developed toward making use of algorithms developed for the Next Release Problem (NRP).

In the NRP, proposed features have an associated cost, so that an optimal feature subset can be selected. Now it might be suggested that the estimated cost of a proposed feature can be used directly in a NRP algorithm. But, there is a practical detail that prevents this: the total cost of a feature includes both the cost of implementation *and* the cost of fixing associated defects.

Of course, defects that will be associated with a feature are not known in advance, so determining their exact cost would be impossible. But if associated defects can be predicted, then the cost of such defects can be estimated.

To this end, a model is proposed for predicting the defects associated with some proposed feature, with some level of confidence. The model will be probabilistic, and will be built using a software project's historical data, available from an issue tracking database.

Exploratory data analysis was conducted on a software project, MongoDB, in order to asses the feasibility of such a predictive model.

The remainder of this document is organized as follows. First, the Background section provides details on the Next Release Problem and the methods used software project data collection. Next, the Related Work section describes other approaches used in defect prediction, which mainly center around static code analysis. Then, the Exploratory Data Analysis section presents the methods used, and results obtained for exploratory data analysis. Finally, the Proposed Work section lays out the specific tasks that are proposed in order to design an effective predictive model, in the context of the results obtained from exploratory data analysis.

2. Background

In this section, first the Next Release Problem will be introduced and described more formally. Then, the methods used for collecting historical data from software projects will be discussed.

2.1. The Next Release Problem (NRP)

The Next Release Problem (NRP) was defined by Bagnall et al[2], and was shown to be NP-Hard. Being abstract in its treatment of feature cost, a broad range of optimization techniques can be applied to the NRP, such as integer programming, hill climbing, simulated annealing, genetic algorithms, etc. The NRP is the subject of academic research in the area of Search-Based Software Engineering[6, 8, 9].

The NRP is designed to aid software project planners, who have multiple customers to satisfy. The project planner would like to maximize the revenue produced from completing the project. This is all described mathematically as follows.

A software project has a set R of all possible requirements (new features and enhancements) that might be included in the next software release. A customer i is satisfied by completing a subset $R_i \subseteq R$. The importance of a customer i is given by the weight, $w_i \in \mathbb{Z}^+$.

Requirements may have acyclic dependencies, or prerequisites, that must be completed first. A subset that includes all prerequisite requirements, recursively, is indicated by $\hat{R}_i$, and should be taken to mean

$$\hat{R}_i = R_i \cup ancestors(R_i) \tag{1}$$

For example, if $R_1 = \{r_2\}$, and r_1 is a prerequisite for r_2, then $\hat{R}_1 = \{r_1, r_2\}$.

A requirement $r \in R$ has a cost $cost(r) \in \mathbb{Z}^+$, associated with its implementation, not considering the cost of any prerequisite requirements. Then, the cost for some subset $R' \subseteq R$ will be

$$cost(R') = \sum \{cost(r) | r \in \hat{R}_i\} \tag{2}$$

Once customer i is satisfied, their weight w_i contributes to the total revenue from the project, as in

$$\sum_{i \in S} w_i \tag{3}$$

So, the NRP is posed as follows: for a group of n customers, select the subset $S \subseteq \{1, 2, ..., n\}$ that maximizes total revenue, while keeping the total cost within some budget constraint B. This is given by

$$\begin{aligned} &maximize && \sum_{i \in S} w_i \\ &subject\ to && cost(\bigcup_{i \in S} \hat{R}_i) \leq B \end{aligned} \tag{4}$$

2.2. Historical Data Collection

The empirical data used to establish a predictive model will be taken from software project historical data, found in an Issue Tracking System (ITS). To facilitate this data collection, only projects that have a freely available ITS will be considered. Included in this category are open-source software projects. Also, because there are many such ITSs, only projects that use the JIRA ITS will be considered.

Data collection will be semi-automatic. First, there is a manual step is to export data from JIRA as XML. Then, pertinent data will be automatically extracted by a Python script and then saved as a delimited table text file for later analysis in R.

3. Related Work

Software defect (bug) prediction typically involves a detailed analysis of code or proposed design changes. Akiyama [1] predicted defect counts based on lines of code (LOC), number of decisions, and the number of subroutine calls. Gafney [3] likewise predicted defect count based on LOC. Rather than code itself, Henry and Kafura [5] define metrics that are based on information taken from design documents, to be used in defect prediction.

TODO: discuss more about other methods

Rather than requiring a detailed code analysis to predict defects, the approach taken in this paper is to look at defect occurrences over time, in order to find a suitably fitting mathematical model. A similar approach is used by Li et al. [7] to fit historical defect occurrences to several mathematical models. Their work considers defect occurrences, looking for trends in regression parameters over consecutive releases. They found that the Weibull model fit best in 73% of the tested software releases. But unlike the approach used by Li et al., which considers only defects by themselves, in this paper the goal is to develop a model for predicting defect occurrence based on the completion of features.

4. Exploratory Data Analysis

To asses the feasibility of developing a predictive model for software defects, exploratory data analysis is performed. This section presents the methods used and discusses the results obtained.

4.1. Data Analysis Methods

Historical data from each software project will be analysed over both the long term (all releases) and short term (each major release). The long-term analysis will provide an overview of each project, and possibly reveal some consistent patterns between projects. The short-term analysis will serve to isolate each major release as a cause-and-effect period (changes being the causes, and bugs being the effects).

Analysis will consist of two parts: descriptive statistics and distribution fitting.

4.2. Results

5. Proposed Work

The Next Release Problem (NRP) is not readily applicable to the release planning process in a software project. This is due to the abstract nature of the NRP. But by modelling the practical details of release planning, the way could be paved for making use of optimization techniques already developed for the NRP.

Feature cost is a critical part of the NRP. An inaccurate cost estimate might mislead a project planner into believing that a feature subset can be implemented inside the budgeted constraint when that may not be the case. And because the cost of fixing defects contributes to the total cost of a feature, the cost of fixing defects should be accounted for.

To this end, the work proposed here is to develop a model for predicting the defects associated with some proposed feature, with some level of confidence. The model will be probabilistic, and will be built using a software project's historical data, available from an issue tracking database.

5.1. Timeline

TODO

A. Software Requirements

The software developed for this paper can be run on any platform that supports Python and R. Besides this basic requirement, here are the other dependencies:

Python dependencies:

- devtools: adds github_install function. Install by running "`install.packages("devtools")`" on the R command line.
- docopt: for defining command-line interfaces and parsers. See the GitHub page for installation instructions.
- BeatifulSoup: for processing XML files. See the support page for installation instructions.

R dependencies:

- docopt: for defining command-line interfaces and parsers. See the GitHub page for installation instructions.

References

[1] F. Akiyama. An example of software system debugging. In *IFIP Congress (1)*, volume 71, pages 353–359, 1971.

[2] A. J. Bagnall, V. J. Rayward-Smith, and I. M. Whittley. The next release problem. *Information and software technology*, 43(14):883–890, 2001.

[3] J. E. Gaffney. Estimating the number of faults in code. *Software Engineering, IEEE Transactions on*, SE-10(4):459–464, July 1984.

[4] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical software engineering and verification*, pages 1–59. Springer, 2012.

[5] S. Henry and D. Kafura. The evaluation of software systems' structure using quantitative software metrics. *Software: Practice and Experience*, 14(6):561–573, 1984.

[6] H. Jiang, J. Zhang, J. Xuan, Z. Ren, and Y. Hu. A hybrid aco algorithm for the next release problem. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*, pages 166–171. IEEE, 2010.

[7] P. L. Li, M. Shaw, J. Herbsleb, B. Ray, and P. Santhanam. Empirical evaluation of defect projection models for widely-deployed production software systems. *SIGSOFT Softw. Eng. Notes*, 29(6):263–272, Oct. 2004.

[8] J. Xuan, H. Jiang, Z. Ren, and Z. Luo. Solving the large scale next release problem with a backbone-based multilevel algorithm. *Software Engineering, IEEE Transactions on*, 38(5):1195–1212, 2012.

[9] Y. Zhang, M. Harman, and S. A. Mansouri. The multi-objective next release problem. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1129–1137. ACM, 2007.