ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 1 January, 2014 Page No.3763-3767

Defect Forecasting In Software System - Mining Approach

Priyank Dineshkumar Patel

Gujarat Technological University, S.P.B. Patel Institute of Technology
Mehsana, Gujarat-384002, India
patel.priyank.d@email.com

Abstract: ***A project's source code change history contains the modification that generates a defect and the change that fixes it. This defect-generation and fix experience can be used to forecast future defects. This research paper presents defect forecasting algorithms that analyze a software project's change history. Defects are related. Our algorithm finds this relation by storing locations that are likely to have defects. That is useful to find most defect prone files. An evaluation of open source projects with more than 5,000 revisions shows that the selected defect training data account for 72%-90% of future defects .By leveraging project history and learning the unique defect patterns of each project, both approaches is used to find locations of defects. This information can be used to increase software quality and reduce software development cost.***

Keywords: Defect, Forecast, VCS, SCM, Mining and Revisions

1. Introduction

Software defects constitute a huge burden for software development firms. The process of finding and removing defects is called debugging, a frequent, tedious, and time-consuming task for software developers. Many defects are not discovered even after an intensive debugging process, with unpredictable and sometimes serious consequences. Current software development practice involves a process of triage on existing uncovered defects, fixing those of highest priority, with large test suites designed to discover latent defects.

1.1 Vision

The purpose of this research is to help developers in the debugging process. To meet this, this work proposes algorithms to locate or predict defects in functions, files, and changes by mining project history data.

The traditional debugging process includes code review, using debugging tools such as GDB [15], regression testing, unit testing, and code review again. Locating and predicting a defect is helpful to developers since it can narrow down the debugging scope. For example, instead of running all test cases and verifying correctness for the entire software project, using the techniques in this dissertation, developers receive lists of likely defective files, methods, or changes. This reduces the scope of software that needs to be examined for defects down to a single file, method, or change. This allows the developer to run just a small set of test cases that are related to the defective locations.

1.2 Challenges

Analysis and Mining of history of open source projects with more than 5,000 revisions. With Mining, separate features and defect related revisions. Store Defect related revisions in db tables and apply algorithm mention in this paper for purpose of forecasting of defect prone files.

On addition of new defect fix in software revisions history, update of training data table using measures describe in this dissertation. And finally find the accuracy of our approach using number of success and fails in forecasting defects.

2. Literature Survey

2.1 Related Work

2.1.1 Statistical Approach [1]

The objective of this paper is to predict software testing defects using statistical models and evaluate the accuracy of the statistical defect prediction model. To determine potential of the statistical models to capture the number of defects on the basis of past data and their metrics, they proceed as follows.

To identify the best predictors in the available set of 18 parameters, they calculate product moment correlation between the number of defects and the predictors. Then they proceed with more advanced statistical models to deal with normal distribution of the target variable and the specifics of the historical data and check multicollinerity within predictor

parameters.

2.1.2 Regression and Arima Hybrid Model Approach [2]

They proposed a multiple linear regression and ARIMA hybrid model for new bug prediction depending upon resolved bugs and other available parameters of the open source software bug report. They analyzed last five year bug report data of an open source software "worldcontrol" to identify the trends followed by various parameters. Bug report data has been categorized on monthly basis and forecast is also on monthly basis. Model accounts for the parameters such as resolved, assigned, reopened, closed and verified bugs respectively.

2.1.3 Sample Based Approach [3]

Throughout development, they proposed sample-based methods for software defect prediction. For a large software system, they selected and tested a small percentage of modules, and then built a defect prediction model to predict defect proneness of the rest of the modules. They described three methods for selecting a sample: random sampling with conventional machine learners, random sampling with a semi-supervised learner and active sampling with active semi-supervised learner.

2.2 Software Configuration Management

Software consists of a collection of items (such as programs, data and documents) that can easily be changed. During software development, the design, code, and even requirements are often changed, and the changes occur at any time during the development. This easily changeable nature of software and the fact that changes often take place require that changes be done in a controlled manner.

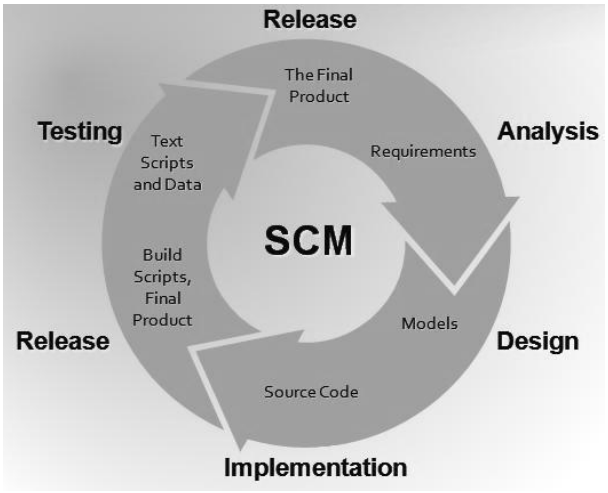

Figure 1: Software Configuration Management life cycle

Software configuration management (SCM) is the discip-line for systematically controlling the changes that take place during development. Software configuration management is a process independent of the development process largely because most development models cannot accommodate change at any time during development.

2.3 Version Control System

Information management systems generally operate on "artifacts." An artifact is an object containing information [6]. A common example of an artifact is a file in a computerized storage system. One class of information management system is a version control system [6]. As each artifact is modified, a new version of the artifact may be saved by the version control system.

For example, a version control system may store files representing source code for a relatively large product, which may be released in multiple revision levels. When one revision of the product is released, the most recent version of some files may have new features that have not been tested or debugged. Accordingly, when that revision of the product is built, prior versions of some files, representing the last version that was fully tested and debugged, may be incorporated into the product. Also, support and maintenance of a revision of the product that was previously released may require access to old versions of a file. Accordingly, many versions of a file may be saved and retrieved for any number of reasons.

A software contributor (developers, testers, designers) works together on common project. Workflow among them defines how contributors of a project work together. There are two workflows available in industry. 1) Centralized and 2) distributed.

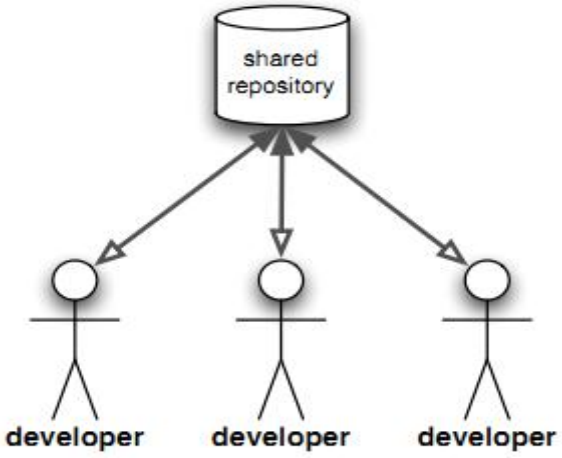

Figure 2: Centralized Version Control System

Artifacts are changed in a working copy and shared with colleagues by committing them to a centralized repository. If the modified artifacts conflict with changes, which occurred in the meantime, they are first merged locally with the latest snapshot from the centralized repository, then committed. Merging changes can be postponed by working on a branch which is merged once the tasks are fulfilled.

When a distributed version control system is used, the members of a project have to agree on a collaboration workflow, to exchange their developments in a structured

order. This collaboration workflow and the applied software development process have to fit to each other, and not every collaboration workflow fits every development process. There is no mechanism in a distributed version control system to notify its users about new versions. To announce a contribution, like a bug fix or a realized feature, mailing lists are used, whereby a contributor posts his contribution along with details how to pull the contributions from him.

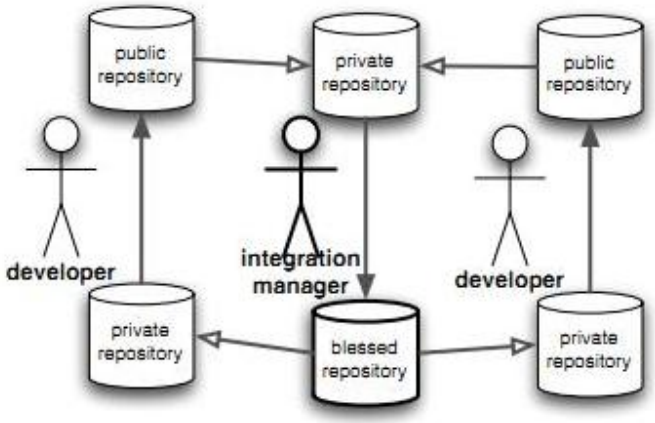

Figure 3: Distributed Version Control System

During the lifetime of a software system, series of changes are made to the software. So many versions will be produced. Version control systems contain significant amounts of data that could be exploited in the study of software evolution [10].

2.4 Defect Tracking System

Paired with SCM, defect-tracking is another key in engineering quality software. Defect-tracking systems guarantee that nothing gets swept under the carpet; they provide a method of Creating, storing, arranging and processing defect reports and enhancement requests. Those who do not use a bug-tracking system tend to rely on shared lists, email, spreadsheets and/or Post-It notes to monitor the status of defects. This procedure is usually error-prone and tends to cause those defects judged least significant by developers to be dropped or ignored [18].

2.5 Terms

2.5.1 Version

A version is a state of an object at one point in the history of its evolution.

2.5.2 Revision

Revision is interchangeable with version, i.e. revision n indicates the nth version. The term revision additionally puts more emphasis on the difference between a version and its previous version. That is, a revision indicates all the changes made to the previous version to generate the current version. The version before the changes is called an old version, and the version after a revision is called the new version.

2.5.3 Hunk

The textual difference of a file between an old version and its new version as computed by a diff tool is represented by a list of text regions, called hunks. A hunk is a series of contiguous changed lines and can be empty. A hunk in the old version is called a deleted hunk, and a hunk in the new version is called an added hunk.

2.5.4 Delta

A deleted hunk always has its corresponding added hunk, and we call a deleted hunk and its corresponding added hunk a delta. There are three kinds of delta: modification, addition, or deletion. If neither the deleted hunk nor the added hunk of a hunk pair is empty, we call this a modification delta. A deletion delta has an empty added hunk, and an addition delta has an empty deleted hunk.

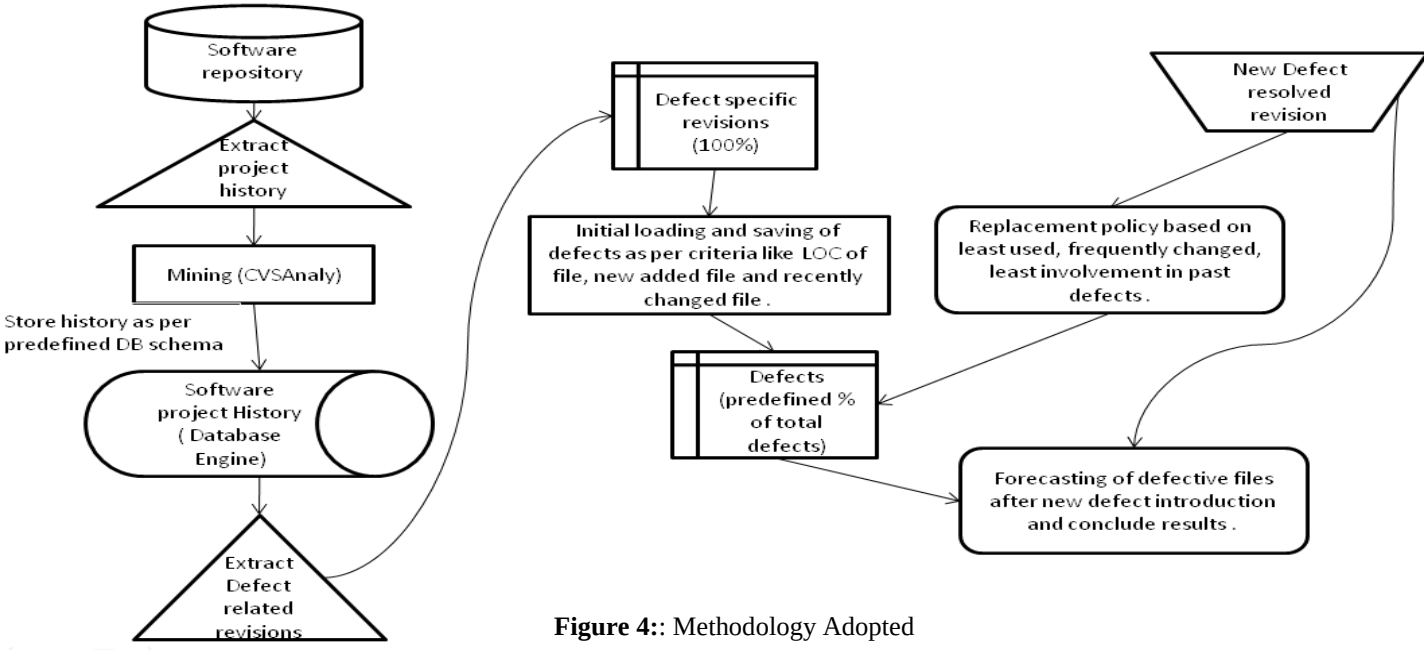

Figure 4:: Methodology Adopted

3. OBJECTIVE AND METHODOLOGY

To assist the analysis, one need to get changes and revisions from the change repository of a project, and perform analysis on every file and change in a revision, we preprocess every revision in the software change repository for a project, extract defect related revisions from it, and transfer the extracted revisions to a database. In this section, we will see the architecture to extract revisions, the steps for pre-processing on the revision data and structure of the database that stores the information.

To analyze software evolution, we use the CVSAnalY tool to extract revisions from SCM repositories [19]. The most commonly used applications to create Source Code Repositories (SCM) at this moment are CVS [20], SVN (Subversions)[23] and GIT[22]. These tools usually register activity and management information in logs. CVSAnalY take these logs, and use them as an input for some basic source code metrics and finally stores the results in a database for posterior analysis. The Architecture for history extraction from such repository is shown in figure 4.

CVSAnalY is responsible for extracting project source versions from a repository of a project under analysis. CVSAnalY automatically checks out the source code of each revision and extracts change information such as the change log message, author, change date, source code, etc.

The extracted source code versions and change information are fed to a pre-processing module, which contains two sub-modules for Software revisions history extraction and defect labeling. The Software revisions history extraction module fetch all revisions , affected files by that revision, Lines of code changes like metric information , commit messages associated with that revision and author, committed timestamp. The defect labeling module differentiates defect fix revisions from non-fix revisions. Finally, the extracted data for the project revisions are saved to the project data database.

4. Analyzed Projects and Observation

We analyzed two open source project and one commercial project written in python. They are Django, Twisted, and CASH respectively.

This all projects are first studied and analyzed for defect pattern. All Three are for different platform. Table 4-1 summarizes all projects we studied, and we briefly describe them as follows.

For this research paper, We have considered SVN and GIT based project As they are widely used in Software industry. Many Large GIT projects and SVN projects are available in github with history. Interval of project history, type of platform, VCS type etc details are mention in Table 1.

Project	Type	Interval	Type of vcs	Open Source (O) / Commercial (C)
Django	Web Framework	07/2005 to 01/2014	GIT	O
Twisted	Networking engine	07/2001 to 01/2014	SVN	O
CASH	Financial Platform	06/2012 to 01/2014	SVN	C

Table 1: Analyzed Projects

With 10% as training data size of defect specific revisions and initiating storing process in training data table taking 5% as per **LOC (lines of code)** , 3% as **newly added files** and 2% as **most recently changed files** , following results for above projects obtained .

Project	No of Revisions	Defect specific revisions	Defect Message pattern	No of revisions in training data table.
Django	24250	15837	Fixed #	1581
Twisted	16784	11868	Fixes #	1183
CASH	657	144	Resolve # , Fixed #	14

Table 2: After Pre-Process and Initialization

5. Results

Now we have training data so when new defect introduced in software system, we have to add this defect revision files in training data set. For that we need replacement policy. We have replaced those defects with criteria like **least used file**, **least frequently changed** and **least involvement** in defects. After replacing, we need to forecast defect. So we checked all files in newly added defect revision against possible affected files in training data.

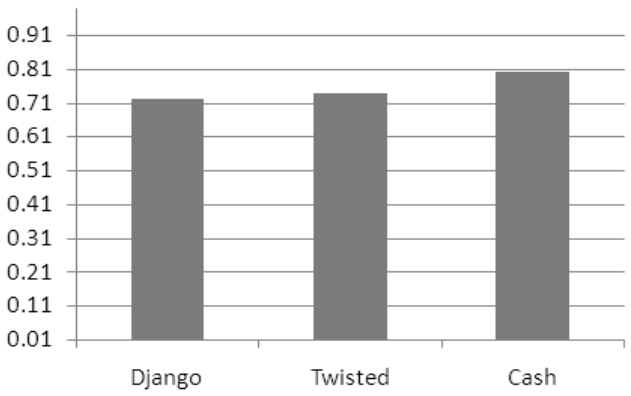

Figure 5: Success Rate for Projects with Different Number of Revisions

If we found the code error or logical error because of that newly added defect then it's termed as success otherwise its termed as fail. After collecting all success and fails for newly added defects, we have analyzed in percentage that how efficiently our algorithm able to forecast defect.

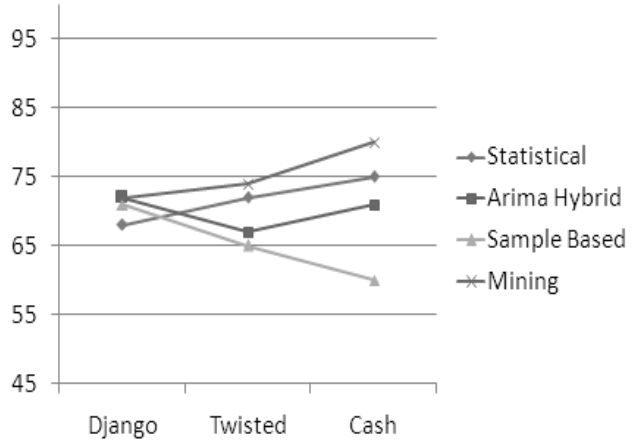

Figure 6: Comparison of Results

Conclusion

Based on this research its concluded that its possible to fore-cast future defects based on past defects introduction and resolve experience. It's also not as complex as Artificial intelligence techniques to which results are compared in results section and its easy to understand, implement and integrate in live projects. It's also concluded by this research that accuracy of future defects forecasting is increasing if few history available of software system. The accuracy can also be increased if more criteria on which algorithm operate is added.

References

[1] Dr Shaik Nafeez Umar. Software Testing Defect Prediction Model – A Practical Approach. International Journal of Research in Engineering and Technology (IJRET). May 2013

[2] Madhur Srivastava, Dharmendra Badal, Ratnesh Kumar Jain(2010), Regression and ARIMA hybrid model for new bug Prediction, International Journal on Computer Science and Engineering, 2(8) , 2622-2628

[3] Ming Li , Hongyu Zhang , Rongxin Wu , Zhi-Hua Zhou, Sample-based software defect prediction with active and semi-supervised learning, Automated Software Engineering, v.19 n.2, June 2012

[4] W.W. Royce. Managing the Development of Large Software Systems. In Proceedings of IEEE WESCON, volume 26. Los Angeles, 1970.

[5] Hanene Cherait and Nora Bounour. Toward a version control system for aspect oriented software. MEDI'11 Proceedings of the First international conference on Model and data engineering, P.110-121, 2011

[6] Christopher Antos, Brian Harry, Thomas McGuire, Justin Pinnix and Michael Sliger. Version control system. Oct -2006.

[7] C. Rodriguez-Bustos and J. Aponte, "How Distributed Version Control Systems Impact Open Source Software Projects", Proc. Ninth IEEE Working Conf. Mining Software Repositories, 2012.

[8] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, P. Devanbu, "The promises and perils of mining git", In Proceeding MSR '09 Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, 2009, 1-7.

[9] Naheed Azeem and Shazia Usmani, "Defect Prediction Leads to High Quality Product", Journal of Software Engineering and Applications, 2011.

[10] Dina Spector, "A NASA Probe Is Spinning Out Of Control After Handlers Lose Contact", Space, http://www.businessinsider.in/A-NASA-Probe-Is-Spinning-Out-Of-Control-After-Handlers-Lose-Contact/articleshow/22471334.cms . 2013.

[11] "Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year" , http://www.prweb.com/releases/2013/1/prweb10298185.htm .2013

[12] "For Want of Hyphen Venus Rocket Is Lost", New York Times, July 27, 1962, as quoted in RISKS Digest, Vol 5, Issue #66.

[13] Tony Long, "Sept. 26, 1983: The Man Who Saved the World by Doing ... Nothing", http://www.wired.com/science/discoveries/news/2007/09/dayintech_0926 . 2007

[14] S. Garfinkel, "History's Worst Software Bugs," 2005, http://wired.com/news/technology/bugs/0,2924,69355,00.html

[15] "GDB: The GNU Project Debugger" , https://www.gnu.org/software/gdb/

[16] J.A. Whittaker. What is software testing? and why is it so hard? Software, 17(1):70{79, 2000.

[17] JD Cem Kaner, E. Hendrickson, and J. Smith-Brock. Managing the Propo of Testers to (Other) Developers. Quality Week, 2001.

[18] The Bugzilla Defect Tracking Tool Guide. http://www.bugzilla.org/docs/2.18/html/ , 2004.

[19] CVSAnalY. https://github.com/MetricsGrimoire/CVSAnalY/ , 2013.

[20] S. Dreilinger, "CVS Version Control for Web Site Projects," 2006, http://cvs.nongnu.org/.

[21] Subversion, "Subversion Project Home Page," 2013, http://subversion.tigris.org/.

[22] Git, "Git Project Home Page," 2013, http://git-scm.com/

[23] SVN, "What is Subversion?", http://svnbook.red-bean.com/en/1.6/svn.intro.whatis.html

[24] Software Engineering: A Practitioner's Approach, 6/e, Roger S Pressman, McGraw Hill, 2005

[25] Software Engineering, Ian Sommerville, 8th Edition, Addison-Wesley, 2006

Author Profile

Priyank Patel is Pursuing M.E. C.S.E. 2nd year**,** received the B.E. degree in Information Technology from Gujarat University in 2011. During 2011-2012, he worked as Web developer in Kraff Software Pvt. Ltd.