ISSN No. 0976-5697

Volume 2, No. 3, May-June 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Study of Importance of Data Mining on Software Repositories

Rituraj Jain*
Department of Computer Engineering
Dharmsinh Desai University
Nadiad – Gujarat, India
jainrituraj@yahoo.com

C. K. Bhensdadia
Department of Computer Engineering
Dharmsinh Desai University
Nadiad – Gujarat, India
ckbhensdadia@ddu.ac.in

Abstract: During all the phases of software development information collected for those phases as well as information generated from them are managed as artifacts. Lots of information about defect, release and source code history is gathered and manage as software repository in large software development. Researchers are trying to find out meaningful information treated as knowledge for the software development by applying data mining methods on these software repositories. This paper highlights the benefits of software repositories as an active repositories used by researchers to gain empirically based understanding of software development, and by software practitioners to predict, plan and understand various aspects of their project.

Keywords: Association Rule Mining, Data Mining, Design Patterns, Mining software Repository, Software Change Repository, Software Development, Software Maintenance, Software Testing.

I. INTRODUCTION

Due to continuous evolution and changes in software systems, it is a challenging task to understand, maintain and enhance them. Managing projects building and maintaining such systems to satisfy the basic software product quality factors i.e. reliable product within the time and budget, is tedious task and it requires use of knowledge build from past experiences.

Many researchers have developed tools, presented methods, and proposed theories to support managers and guide developers as they evolve large software systems [1], [2].

It is always good that group of individuals work effectively as a team to develop large scale software. All the communications made in this collaborative work are manage as documents in software repositories, which are shared among each other. Repositories built during software evolution possess wealth of valuable information regarding the evolutionary history of a software project and can be used by developers to manage their project [3], [4].

II. EXAMPLE OF SOFTWARE REPOSITORIES

Source control repositories, bug repositories, archived communications, deployment logs, and code repositories are examples of software repositories that are commonly available for most software projects.

Source control repositories contain development history of a project like changes made to the source code, name of the developers who is responsible for those changes, the time on which changes were made, other side effects due to these changes and describing massages for all the changes. CVS, subversion, Perforce, ClearCase are examples of source control repositories. Bug find in the projects and how they are resolved are maintained in Bug repositories as bug reports. It also tracks the additional feature requests that are reported by users and developers of large software system. Bugzilla and Jira are examples of bug repositories. Discussions made by the different stake holders about software project throughout its lifetime are archived in Archived communications repositories. Mailing lists, emails, IRC chats, and instant messages are examples of archived communications about a project. Onsite installation history of software product as deployment is recorded in Deployment logs repositories. Code repositories archive the source code for a large number of projects. Sourceforge.net and Google code are examples of large code repositories [4].

Many researchers have shown the usefulness of software repositories during software development. It can be used to identify hidden code dependencies suggested by Gall et al. [6], can support management in building reliable software systems by predicting bugs and effort suggested by Avaya, Graves et al. [7] and Mockuset et al. [8] and can assist developers in understanding large systems suggested Chen et al. [5]

III. MINING ON SOFTWARE REPOSITORIES

Developers can used above mentioned repositories by applying mining techniques on them to automate and improve the extraction of information to gain knowledge for manage their projects.

A. *Mining Support for Software Requirements*

Capturing the traceability information always backed by software engineering and software assurance but it is still taking part in to a big discussion and not used meaningfully in software development due to the constraints of time and error prone activity. Software development team can get improve the quality of requirements by requirements assurance and assurance of the traceability matrix (TM). Finding the complete and correct relationships and dependencies between all requirement set (functional or non-functional) is again time consuming and error prone activity for assurance. Port, D. et al. [8] used text-mining and statistical methods to reduce this effort and increase TM assurance. Their method used mixed model of similarity and dissimilarity in requirements to generate trial sets which help to identify low risk and problem area associated with Non functional Requirements – Functional Requirements - Requirement Traceability Metrics.

B. *Mining in Software Design and Coding*

Architectural design is a base to get quality in software systems. Experiences of experts from past projects and practices are captured as design patterns. Design patterns are micro architectures that have proved to be reliable, easy-to implement and robust. Design patterns can be use as supportive knowledge in similar type problems where expert can reuse their experiences in software system design. Mining design pattern instances from system design and source code can greatly help to understand the systems and change them in the future. It also helps to trace back to the original architectural design decisions which are generally lost in system source code [14] [15] [16].

Z. Zhang, et al. [15] proposed the concept to describe structures (mainly the class diagram) of system design and design pattern by using extended graph. Extended graph are used to describe class diagrams, where the vertices can be mapped onto the classes. The instances of design pattern can be discovered by finding the sub-graph isomorphism. Extended graphs represent relations more clearly by handling the inheritance and aggregation relations and also they are easier to manipulate.

Basu, N. et al. [16] discovered design patterns in the source code which provides information regarding how the patterns work by describing basic object oriented structural information. They define the notion of Extensible Pattern Markup Language (XPML) to describe mined design patterns. Their process started from analysis of source code using reverse engineering engine (REE) and build semantic graph in XML is followed by loading of XPML pattern description file into a standard XMLDOM tree and compared with the Semantic Graph. They search the candidate classes by filtering the source classes with appropriate properties and then selected source classes bound with pattern classes. Selected classes have been tested with all the possible combination to see if they form a pattern. Found combinations were checked again to see that they possess call delegations, object creations and whether they redefine/override the appropriate operations or not. Bound source class come through the last check were define as design patterns instance.

Maqbool, O. et al. [17] use data mining for software reverse engineering by applying association rule mining algorithm and using tools on source files of a software system to gain knowledge about that software system. Their method comprises of two steps: item selection and identification of association rule on these items. They used functions and global variables along with used defined type as items. Their results show that by using association rule mining to find interesting association between functions, types and global variables within the source files one can gain deeper understanding of the code, and may be used to restructure the code for maintainability. In some cases, the associations found can be helpful in remodularizing the code, e.g. in converting a structured design to an object-oriented design.

C. *Mining in Software Testing*

Test case design in Software testing is an art and the design of software tests is mostly based on the testers' expertise. Manual approaches to software testing require a complete knowledge of the facts collected as requirements during requirement analysis. Mark Last et al. [11] applied mining algorithms called Info-Fuzzy Network (IFN) for automated induction of functional requirements from execution data. Their algorithm is trained on inputs provided by Random Tests Generator (RTG) and outputs obtained from a legacy system. IFN algorithm works on the training given by RTG module and the outputs produced by the Legacy System for each test case and the descriptions of variables from Specification of Application Inputs and Outputs module. The generated data mining models can be utilized for recovering specifications, regression tests, and for testing new system to evaluating the correctness of software outputs.

D. *Mining in Software Maintenace*

Software maintenance is the final phase of software development. After the product is releases primary modification whenever is required in the product will be done through this phase only. Modifications are required due to resolve the faults found after delivery, to addition of new product attributes, or to give support with new changing environment.

From total budget of the any software product development maximum cost is incurred on removing the errors in maintenance phase only. Meanwhile the system is unavailable and it will lead to increase the cost of maintenance. To come with models and the factors which affect the outcome and quality of software maintenance tasks, so many researches are going on. Uzma Raja, Marietta J. Tretter [13] defined a model using Data Mining (DM) techniques applied on maintenance data of Open Source Software and identifies the factors affecting maintenance outcomes. They measure the maintenance quality of an OSS project by Mean Time to Repair (MTTR) factor of an error. Its low value suggests that developing team and product is of good quality while its high value suggests that team is not responding quickly to errors. Text miner they used to categorized project into different clusters. By Combining cluster information with other variables they build a model using Data Mining. The model was evaluated performing diagnostic testing and Logistics Regression evaluation criteria. They suggested some factors like errors reported by users, number of download have a positive impact while use of mail and age of the project have a negative impact on maintenance quality. They also suggest that projects having higher users' interactions for error identification have a greater probability of having low MTTR and high quality. Number of downloads can be used as an indicator of the popularity of the project.

E. *Mining on Software Change Repositories*

The revision history of a software system shows how and why the system evolved in time and who is responsible for the new changes. The revision history can also show relationship exists between the parts of the software system. So the knowledge gained by applying mining to version histories guide programmers for change propagation. It suggests and predicts likely further changes required due to changes done earlier, shows item coupling which is actually undetectable by program analysis, and can prevent errors due to incomplete changes.

Ying et al. developed an approach that uses association rule mining on CVS version archives [1] to find out change patterns. Their method finds dependencies between the codes and assists the developer for co changes required. In their approach they applied association rule mining on the data items selected form software configuration management. Change patterns generated from the rule mining suggest the file in which the modification is required.

Xing and Stroulia [9] also propose a method for recovering the relationships between the system classes by applying apriori association rule mining on versions of UML diagrams to detect class co-evolution. They define UMLDiff algorithm which takes two UML class models represented in XMI as input and parsing the input into two labeled tree structures and identify difference between the two tree structures. The results are represented as change trees in an XML-based syntax.

Hassan and Holt [10] proposed a model to investigated heuristics to predict change propagation. Their model repeatedly works onto identify the changes requires base d on the initial entity for change suggested by developers. The goal of change propagation is to ensure the consistency of assumptions among these interdependent entities. Model define by them can also use expertise of senior developer, or software development tool, or even a suite of tests. Based on this model they concluded heuristics for predicting change propagation few of them are Developer Based Co-changes (DEV), Entity Based Historical Co-change (HIS), Entity Based Code Structure using Call, Use, and Define (CUD), and Entity Based Code Structure using Code Layout (FIL).

Sunghun Kim et al. [13] have been define change classification technique to locate the latent software bugs in changes and classify them as a buggy change or clean change. Their classifier is trained using features extracted by using the information collected from configuration management repository. They suggest that their method helps developers by prompting prediction of buggy changes when editing of the source code is going on or right after a change submission.

Huzefa Kagdi et al. [12] presented a method to recover/discover traceability links between software artifacts. They applied sequential-pattern mining on software repositories and identified the set of artifacts having dependencies in changes in specific order which is basically generates traceability links' direction. This order is They applied their approach on open-source software system KDE (K Desktop Environment) and uncovered traceability links between various types of software artifacts (e.g., source code files, change logs, user documentation, and build files) with high accuracy.

IV. CONCLUSION

Software repositories could be mined to uncover useful patterns and actionable information about software systems and projects. So many researchers have defined theories and techniques for mining the different repositories in order to solve important and challenging problems, such as identifying bugs, and reusing code, which practitioners must face and solve on a daily basis.

V. REFERENCES

[1]. Annie T.T. Ying, Gail C. Murphy, Raymond Ng, Mark C. Chu-Carroll, "Predicting Source Code Changes by Mining Change History" IEEE Transactions on Software Engineering, vol. 30, no. 9, pp. 574-586, Sept. 2004, doi:10.1109/TSE.2004.52

[2]. Ahmed E. Hassan, "Mining Software Repositories to Assist Developers and Support Managers" ICSM, pp.339-342, 22nd IEEE International Conference on Software Maintenance (ICSM'06), 2006

[3]. Prasanth Anbalagan, Mladen Vouk, "On mining data across software repositories" MSR, pp.171-174, 2009 6th IEEE International Working Conference on Mining Software Repositories, 2009

[4]. A. E. Hassan. "The road ahead for mining software repositories" In Proceedings ICSM, FoSM track, pages 48--57, 2008.

[5]. Annie Chen, Eric Chou, Joshua Wong, Andrew Y. Yao, Qing Zhang, Shao Zhang, Amir Michail, "CVSSearch: Searching through Source Code using CVS Comments" ICSM, pp.364, 17th IEEE International Conference on Software Maintenance (ICSM'01), 2001

[6]. Harald Gall, Karin Hajek, Mehdi Jazayeri, "Detection of Logical Coupling Based on Product Release History" ICSM, pp.190, 14th IEEE International Conference on Software Maintenance (ICSM'98), 1998

[7]. Todd L. Graves, Alan F. Karr, J.s. Marron, Harvey Siy, "Predicting Fault Incidence Using Software Change History" IEEE Transactions on Software Engineering, vol. 26, no. 7, pp. 653-661, July 2000, doi:10.1109/32.859533

[8]. Port, D.; Nikora, A.; Hayes, J.H.; LiGuo Huang, "Text Mining Support for Software Requirements: Traceability Assurance" Proceedings of 43rd IEEE Hawaii International Conference on System Sciences (HICSS 2011), pp. 1-11

[9]. Z. Xing and E. Stroulia, "Data-Mining in Support of Detecting Class Co-Evolution" Proceedings 16th International Conference on Software Engineering and Knowledge Engineering SEKE '04, pp. 123-128.

[10]. Ahmed E. Hassan, Richard C. Holt, "Predicting Change Propagation in Software Systems" ICSM, pp.284-293, 20th IEEE International Conference on Software Maintenance (ICSM'04), 2004

[11]. Mark Last , Menahem Friedman , Abraham Kandel, "The data mining approach to automated software testing" Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, August 24-27, 2003, Washington, D.C. [doi>10.1145/956750.956795]

[12]. Huzefa Kagdi, Jonathan I. Maletic, Bonita Sharif, "Mining Software Repositories for Traceability Links" ICPC, pp.145-154, 15th IEEE International Conference on Program Comprehension (ICPC '07), 2007

[13]. Sunghun Kim, E. James Whitehead, Jr., Yi Zhang, "Classifying Software Changes: Clean or Buggy?" IEEE Transactions on Software Engineering, vol. 34, no. 2, pp. 181-196, Mar./Apr. 2008, doi:10.1109/TSE.2007.70773

[14]. J. Dong, Y. Zhao, and T. Peng, "A review of design pattern mining techniques", International Journal of Software Engineering and Knowledge Engineering *(*IJSEKE*)*, 2008.

[15]. Z. Zhang, Q. Li, and K. Ben, "A new method for design pattern mining" Proceedings of the 3rd International Conference on Machine Learning and Cybernetics, 2004

[16]. Basu, N. Chatterjee, S. Chaki, N. "Design Pattern Mining from Source Code for Reverse Engineering" TENCON 2005 pp. 1 - 7 IEEE Region 10 Conference.

[17]. Maqbool, O.; Karim, A.; Babri, H.A.; Sarwar, M. **"**Reverse Engineering Using Association Rules" In proceedings of 8th IEEE International Multitopic Conference - INMIC 2004, pp. 389 - 395