Volume 2, No. 2, Mar-Apr 2011
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Analysis of Mining Techniques for Version Histories to guide software changes

Dr Gurdev Singh
Department of Computer Science & Engg.
Amritsar,Punjab
hodcsegimet@gmail.com

Er Babaldeep Kaur
Department of Computer Science & Engg.
Amritsar,Punjab
babbaldeep@yahoo.co.in

Er Gagandeep Singh*
Department of Computer Science & Engg.
Amritsar,Punjab
gagandeep.engineer@gmail.com

Abstract: The histories of software systems hold useful information when reasoning about the systems at hand or about general laws of software evolution. Modern software has evolved to meet the need of stakeholders, but the nature and scope of this evolution based on mining version histories is difficult to anticipate and manage. In this paper we examine techniques which can discover interesting patterns based on mining using association rules and training the network that can guide software developers about the changes. Mining the version histories of software suggest and predict further likely changes and can prevent errors due to incomplete changes and provide an edge in software evolution.

Keywords: Association rule, Software Evolution, Neural Network

I. INTRODUCTION

The importance of observing and modeling software evolution started to be recognized in 1970's with the work of Lehman [1]. Since then more and more research has been spent to identifying the driving forces of software evolution, and to using this information to better understand software. Before going into details, we define three terms: version, evolution and history. A *version* is a snapshot of an entity at a particular moment in time. The evolution is the process that leads from one version to another. A history as the reification which encapsulates knowledge about evolution and version information. According to these definitions, we say that we use the history to understand the evolution. Suppose you are a programmer and just made a change. What else do you have to change? In earlier work, researchers have used history data to understand programs and their evolution [2].In this work we will discuss the ROSE tool to leverage version histories. In contrast to present work our research work

1 uses data mining techniques to obtain association rules from version histories
2 detects coupling between fine-grained program entities such as functions or variables/functions.

II. PROCESSING THE DATA

Figure 1. Shows the Rose server. The *ROSE server* reads a version archive groups the changes into *transactions,* mines the transactions for *rules* which describe implications between software entities "If akeys[] is changed, then initDefaults() is changed, too." When the user changes some entity (say, aKeys[]),

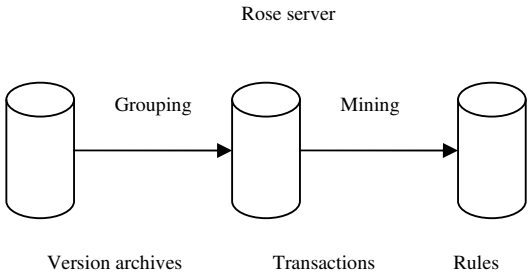


Figure 1

the *ROSE client* queries the rule set for applicable rules and makes appropriate suggestions for further changes.(figure 2)

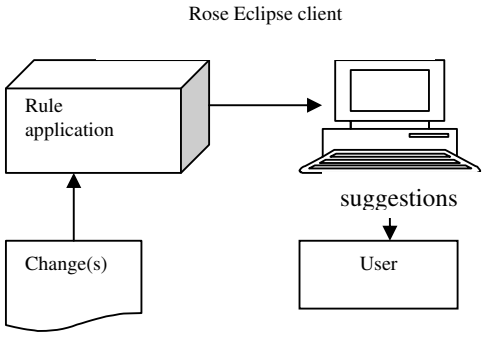


Figure 2

III. RELATING THE CHANGES

Most modern version control systems have a concept of product versioning—that is, one is able to access transactions as they alter the entire product. CVS, though provides only *file versioning*. To recover per-product

transactions from CVS archives, we must *group* the individual per-file changes into individual transactions. CVS has no syntactic knowledge about the files it stores; it manages only files and line numbers for each change. ROSE thus *parses* the files; associating syntactic entities with line ranges, ROSE can thus relate any change (given by file and line) to the affected components.

IV. TRANSACTION TO RULES

Given the transactions as described in the previous sections, the aim of the ROSE server is to mine *rules* from these transactions. What is a rule? Here is an example:
{(rg.java, *field*, a*Keys*[])}
) { (rg.java, *method*, *initDefaults*()),
(plug.properties, *file*, plug.properties) }

This rule means that whenever the user changes the field a*Keys*[] in rg.java, then she *should* also change the method *initDefaults*() and the file plug.properties.

Support. The support determines the *number* of transactions the rule has been derived from. Assume that the field a*Keys*[] was changed in 8 transactions. Of these 8 transactions, 7 also included changes of both
the method *initDefaults*() and the file plug.properties. Then, the *support* for the above rule is 7.

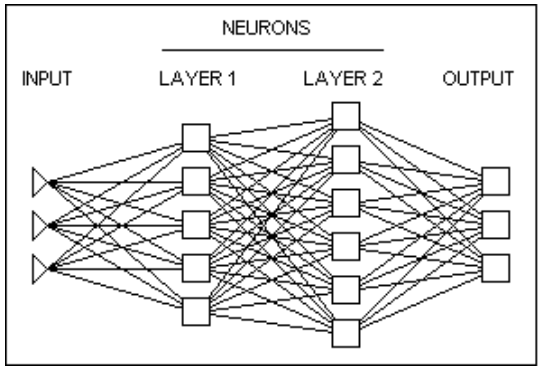
Confidence. The confidence determines the *strength* of the consequence, or the relative amount of the given consequences across all alternatives. In the above example, the consequence of changing *initDefaults*() and plug.properties applies in 7 out of the 8 transactions involving *fKeys*[]. Hence, the *confidence* for the above
rule is 7/8 = 0.875.

A. Association rule

ROSE uses the Apriori Algorithm to compute association rules.

V. NEURAL NETWORK

A neural network is first and foremost a graph, with patterns represented in terms of numerical values attached to the nodes of the graph and transformations between patterns achieved via simple message-passing algorithms. Certain of the nodes in the graph are generally distinguished as being input nodes or output nodes, and the graph as a whole can be viewed as a representation of a multivariate function linking inputs to outputs. Numerical values (*weights*) are attached to the links of the
graph, parameterize the input/output function and allowing it to be adjusted via a learning algorithm.



The neural network approach can be used for mapping transactions to rules. for example if a change in akeys[] field affects the initdefault[] every time. Then neural network can be traine for such things. i.e Training the network in such a way that transaction to rules will be based on historical data changes in the different versions. if there is change in one module effects the other modules that are coupled , this can be done through neural network.

VI. RESULT

In particular, our evaluation does not allow any conclusions about the predictive power for closed-source projects, as Rose is an open source tool. Transactions do not record the *order* of the individual changes involved. Hence, our evaluation cannot take the order into account the changes were made—and treats all changes equal. In practice, we expect specific orderings of changes to be more frequent than others, which may affect results for navigation and prevention.

VII. RELATED WORK

Gall et al. were the first to use release data to detect logical coupling between modules [4] The CVS history allows detecting more fine-grained logical coupling between classes [5], files and functions [6].

VIII. CONCLUSION

ROSE can be a helpful tool in suggesting further changes to be made, and in warning about missing changes .Neural network approach can make the system fast as Batch training of a network proceeds by making weight and bias changes based on an entire set (batch) of input vectors. Incremental training changes the weights and biases of a network as needed after presentation of each individual input vector. Incremental training is sometimes referred to as “on line” or “adaptive” training. Neural networks have been trained to perform complex functions.

IX. RERERENCES

[1] Manny M. Lehman, and Les Belady. Program Evolution Processes of Software Change. London Academic Press, 1985.

[2] T. Ball , J.-M. Kim, A. A. Porter and H. P. Siy. “If your version control system could talk.”. .In *ICSE*

Workshop on Process Modelling and Empirical Studies of Software Engineering, 1997.

[3] BISHOP, C. M. 1995, "*Neural Networks for Pattern Recognition"*. Oxford University Press, New York.

[4] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history". In *Proc. International Conference*

on Software Maintenance *(ICSM '98)*, pages 190–198, Washington D.C., USA, IEEE

[5] H. Gall, M. Jazayeri, and J. Krajewski , "CVS release history data for detecting logical couplings". IWPSE 2003 [15], pp 13–23.

[6] T. Zimmermann, S. Diehl and A. Zeller. "How history justifies system architecture (or not)." In IWPSE 2003, pp 73–83.

[7] Filip Van Rysselberghe, and Serge Demeyer, "Studying software evolution information by visualizing the change history". In *Proceedings of The 20th IEEE International Conference on Software Maintenance (ICSM 2004)*, 2004.

[8] Thomas Zimmermann, P. Wegerber, S. Diehl, and Andreas Zeller, "Mining version histories to guide software changes". In *26th International Conference on Software Engineering (ICSE 2004)*, pp 563–572, 2004.

[9] M. Morisio, M. Ezran, and C. Tully, "Success and Failure Factors in Software Reuse," IEEE Trans. Software Eng., vol. 28, no. 4, pp. 340-357, Apr. 2002

[10 R. Robbes, "Mining a change-based software repository," in Proceedings of the 4th International Workshop on Mining Software Repositories (MSR 2007). ACM Press, 2007, p. 15.

[11] R. Robbes and M. Lanza, "Versioning systems for evolution research," in Proceedings of IWPSE 2005 (8th International Workshop on Principles of Software Evolution). IEEE CS Press, 2005, pp. 155–164.

[12] L. Yu and S. Ramaswamy, "Mining cvs repositories to understand open-source project developer roles," in MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories. Washington, DC, USA: IEEE Computer Society, 2007, p. 8.

