On the Transfer of Evolutionary Couplings to Industry[1]

Piërre van de Laar
Embedded Systems Institute
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
pierre.van.de.laar@esi.nl

Abstract

In this paper, we describe a case study at Philips Healthcare MRI focusing on evolutionary couplings, i.e., a technique to infer relationships among modules by analyzing their history of changes in the source code archive. In this case study, we failed to transfer CouplingViewer, a tool implementing the current state-of-art in evolutionary couplings, to industry. According to the industrial experts an important industrial requirement was not met: the signal-to-noise ratio was too low.

1. Introduction

Evolving a software product requires active management of the couplings between the modules [1]. Low coupling between modules not only makes the software easier to understand, but also minimizes the paths along which changes and errors can propagate [2]. One popular approach to reconstruct couplings between modules is based on mining the change history [3-17]. Like [9], we will use the term *evolutionary couplings* to refer to this approach. Evolutionary coupling analysis is based on the conceptual model [18] that a change is made for a specific reason and the modules changed are thus semantically related [3]. Although evolutionary coupling analysis has been applied on large scale industrial and open source software, to our knowledge, it has not been transferred to industry, i.e., being used by multiple developers and architects without support of a group of researchers.

In this paper, we describe a case study carried out as part of the Darwin project [19] at Philips Healthcare MRI. In this case study, we tried to transfer tooling for evolutionary coupling analysis to industry. In the industrial toolbox, many "tools" are available to detect and visualize specific couplings, ranging from include relations to call graphs, and from documentation to inter-process communication monitoring. Although each tool has its limitations, all tools together constitute a powerful industrial toolbox. Therefore, we expected the bar of acceptance of evolutionary couplings tooling to be high. In the case study, we would like to get answers to the following questions:

- What are the industrial requirements for applying evolutionary couplings?
- Are these requirements already met by the current state-of-art?

The paper is organized as follows. In Sections 2 and 3, we discuss the related work. Section 2 focuses on what is a change, and where can we get information about changes. Section 3 focuses on the visualisation of the information about changes. In Section 4, we describe the case study in which we attempted to transfer evolutionary couplings to the industrial context of Philips Healthcare MRI and the lessons learned. We end in Sections 5 and 6 with the discussion and conclusion.

2. Changes

The software of most systems is developed, maintained, and evolved in many iterations. In each iteration, the software is changed, e.g., to add a feature, fix a bug, or improve its performance. At the end of each iteration, the change is accepted and the software is thus in an acceptable state. Typically, this means that the software compiles and the acceptance tests have been successfully executed. Iterations can be observed at different levels of abstraction. For example, an iteration between product releases contains many smaller iterations. Evolutionary coupling analysis has

[1] This work has been carried out as part of the DARWIN project at Philips Healthcare under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the BSIK program.

been applied with iterations at different levels of abstraction [3][4][10][11].

For evolutionary couplings, the change sets, i.e., the set of modules modified together in a change, are needed. This information might be captured by the version control system, as is done by subversion [20], or in commit mails [11]. If this information is not captured, as is the case with the version control system CVS, it must be approximated. Time windows are a good approximation for restoring change sets from CVS [11].

Often, the information contained in repositories needs preprocessing (see e.g. [11]), since the conceptual models [18] of these repositories do not exactly match the conceptual model of evolutionary couplings.

3. Visualization of changes

One way to visualize the changes to a software archive is a module-change matrix. Each row represents a single change and indicates which modules were changed, i.e., which modules were part of that change. For a successful, long-living system the number of changes becomes so large that the module-change matrix becomes inconvenient to be handled by humans. This raises the question: How could this information be effectively summarized? In the literature, a number of different similarity measures between the different entities, i.e., modules, have been proposed to answer this question [3-5][12-15][21].

Summarizing the information of all changes into change similarity values between all entities in the system still poses a visualization issue for industrial systems, which contain tens of thousands of files, hundreds of components, and tens of subsystems. The hierarchical structure in the software can be exploited to limit the number of visible entities to a manageable number [22], while ensuring by using navigation along the hierarchical structure that all entities could still be made visible.

4. Case Study at Philips Healthcare MRI

Within the repository of Philips Healthcare MRI, the information of changes is captured in so-called postlists. This information includes an explanation message describing the change; and the set of files that were modified to realize the change. A change described in a postlist is reviewed. When the reviewers accept the change, the printed version of the postlist of the change is signed, as required by the Food and Drug Administration (FDA), and submitted to the integrator. Once per day, the integrator applies all submitted postlists to the archive and performs the integration tests. When the integration tests are executed successfully, the set of postlists is accepted and made available to all developers. Postlists thus have a controlled commit, similar to many open source projects, where changes are analyzed and discussed by its project members over newsgroups, email, and mail lists before they are committed, see, for example, [16] and the references therein. Controlled commits reduce the probability of incomplete changes, and of combining unrelated changes [16].

Some differences in the conceptual models of postlists and evolutionary couplings exist. We dealt with merges and changes in the file structure as is done in literature.

4.1. CouplingViewer

We based CouplingViewer, our tool to visualize evolutionary couplings, on postlists. Postlists are, like commit mails, a more precise solution than approximating change sets using time windows [11]. Figure 1 shows a screenshot of CouplingViewer.

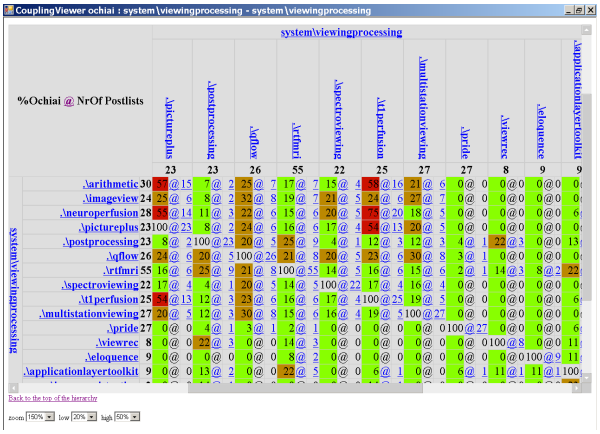

Figure 1. Screenshot of CouplingViewer showing the evolutionary couplings between elements in the subsystem *viewingprocessing*.

We followed the literature in which all except one [13] use symmetric similarity measurements. We decided to use both an absolute and a relative similarity measurement to have the best of both worlds: Absolute measures are dependent on the way of working by the development organization and the developers [16]. Relative measures give unreliable results for rarely changing entities, since a large amount of data is needed before the measure can be accurate [17]. In

particular, CouplingViewer presents both the number of co-changes and the Ochiai percentage [3][21].

Finally, we defined that a hierarchical entity changes when any of the entities it contains changes. For a file system, this means that, according to this definition, a directory changes when at least one of the files or directories contained in that directory changes. Since changes are aggregated to all higher hierarchical entities, guidance to navigate to often changing entities in the complete hierarchy is automatically provided.

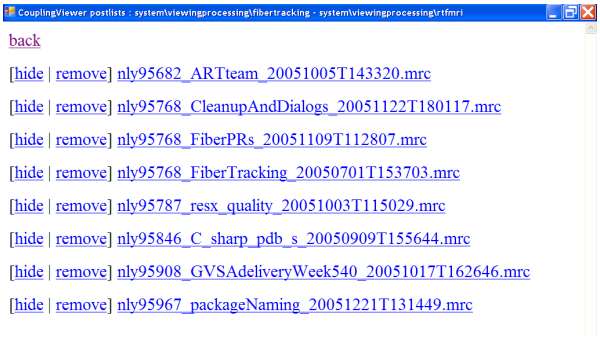

Figure 2: Part of the list of changes in which two entities were changed together.

Besides the navigation within the hierarchy of the software, we also added navigation to zoom into the "reason" of the evolutionary couplings. Each value, representing the number of co-changes, provides a navigational link to the list of changes in which these two entities were changed together, as depicted in Figure 2. In this list, the navigational link associated with each change, enables one to navigate to the information of this change, for example, to read its explanation message that elaborates on the reason of that particular change. The information about the reason of an evolutionary coupling is relevant from a scientific and industrial point of view. From the scientific point of view, the reason is needed to verify whether the evolutionary coupling represents a real coupling [4]. From an industrial point of view, the reason of an undesirable coupling is needed to be able to improve the structure of the software by removing that coupling.

When the information of a change indicated, according to the user, that the change was not made for a single reason, the user could, at run-time, *hide* this change in the current session, or even *remove* it from the current and all future sessions, by clicking the respective link in front of the change, as shown in Figure 2.

4.2. Industrial validation

4.2.1. Pilot study. Approximately 20 software developers and architects have experimented with CouplingViewer in a pilot study. They investigated couplings found by mining the change history that were unexpected. Furthermore, they investigated whether known couplings were also found by mining the change history using CouplingViewer. Based on the experiments, these software developers and architects did not want to add CouplingViewer to their industrial toolbox. According to them, an important industrial requirement was not met: the signal-to-noise ratio was considered too low for industrial applicability. In other words, too many false positives and false negatives were observed in the evolutionary couplings.

4.2.2. Quantitative experiment. To quantify the observed signal-to-noise ratio, we set up an experiment in which four software developers/architects had to determine whether 20 pairs of software entities were coupled. During the experiment, each software developer/architect had to answer in 90 minutes the following two questions for each of the 20 pairs of software entities:

- According to CouplingViewer, is there a coupling between this pair of software entities?
- Looking at the evidence, such as changes described in postlists, source code, documentation, experience, etc., is CouplingViewer right about the coupling between this pair of software entities?

The outcome of the experiment is that in 15% of all cases in which CouplingViewer reports an evolutionary coupling, no actual coupling exists; and in at least 6.3% of all cases in which CouplingViewer reports no evolutionary coupling, an actual coupling does exist.

5. Discussion

To our knowledge, previous studies [3-17] did not report the number of false positives and false negatives that were observed in their case studies, although examples of false positives and false negatives were given [11][13]. Despite the fact that we are unable to compare performances, we believe that our performance is comparable to the current state-of-art performance in evolutionary coupling analysis.

6. Conclusions

In this paper, we describe a case study at Philips Healthcare MRI focusing on evolutionary couplings, i.e., the reconstruction of the couplings between modules by mining the change history. In this case study, we tried to transfer CouplingViewer, a tool implementing the current state-of-art in evolutionary couplings, to industry. CouplingViewer had an accuracy of 15% false positives and at least 6.3% false negatives in the evolutionary couplings. However, according to the industrial experts, an important industrial requirement was not met: the signal-to-noise ratio was too low. In the future, we would like to investigate what the signal-to-noise ratio should be for industrial applicability; what the sources of noise are and whether this noise can be prevented; and we would like to focus on the threats of validity of the experiment to determine whether our failure to transfer CouplingViewer is only caused by a too low signal-to-noise ratio.

7. Acknowledgements

I would like to thank the software architects and developers of Philips Healthcare MRI that participated in evaluating CouplingViewer. Furthermore, I would like to thank Gerrit Muller, Pierre America, Trosky Callo, Richard Doornbos, Adam Vanya, Pieter van der Spek, and the anonymous reviewers for useful feedback on an earlier version of this paper.

8. References

[1] M.M. Lehman, J.F. Ramil, P.D. Wernick, D.E. Perry, and W.M. Turski, *Metrics and Laws of Software Evolution - The Nineties View*, Proceedings 4th International Symposium on Software Metrics, 1997, pp. 20-32.

[2] W.P. Stevens, G.J. Myers, and L.L. Constantine, *Structured Design*, IBM Systems Journal, Vol. 13(2), 1974, pp. 115-139.

[3] T. Ball, J.-M. Kim, A.A. Porter, and H.P. Siy, *If Your Version Control System Could Talk …*, Proceedings ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering, 1997.

[4] H. Gall, K. Hajek, and M. Jazayeri, *Detection of logical coupling based on product release history*, Proceedings International Conference on Software Maintenance, 1998, pp. 190-198.

[5] H. Gall, M. Jazayeri, and J. Krajewski, *CVS Release History Data for Detecting Logical Couplings*, Proceedings 6th International Workshop on Principles of Software Evolution, 2003, pp. 13-23.

[6] A.T.T. Ying, G.C. Murphy, R. Ng, and M.C. Chu-Carroll, *Predicting Source Code Changes by Mining Change History*, IEEE Transactions on Software Engineering, Vol. 30(9), 2004, pp. 574-586.

[7] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, *Mining Version Histories to Guide Software Changes*, IEEE Transactions on Software Engineering, Vol. 31(6), 2005, pp. 429-445.

[8] A. Vanya, L. Hofland, S. Klusener, P. van de Laar, and H. van Vliet, *Assessing Software Archives with Evolutionary Clusters*, Proceedings 16th IEEE International Conference on Program Comprehension, 2008, pp. 192-201.

[9] H. Kagdi, M.L. Collard, and J.I. Maletic, *A survey and taxonomy of approaches for mining software repositories in the context of software evolution*, Journal of Software Maintenance and Evolution: Research and Practice, Vol. 19, 2007, pp. 77-131.

[10] M. D'Ambros, H.C. Gall, M. Lanza, and M. Pinzger, *Analysing Software Respositories to Understand Software Evolution*, Chapter 3 of T. Mens and S. Demeyer (Eds.), *Software Evolution*, 2008, pp. 37-67.

[11] T. Zimmermann and P. Weißgerber, *Preprocessing CVS Data for Fine-Grained Analysis*, Proceedings International Workshop on Mining Software Repositories, 2004, pp. 2-6.

[12] G. Antoniol, V.R. Rollo, and G. Venturi, *Detecting groups of co-changing files in CVS respositories*, Proceedings 8th International Workshop on Principles of Software Evolution (IWPSE), 2005, pp. 23-32.

[13] M. Burch, S. Diehl, and P. Weißgerber, *Visual Data Mining in Software Archives*, Proceedings ACM Symposium on Software Visualization, 2005, pp. 37-46.

[14] J. Ratzinger, M. Fisher, and H. Gall, *EvoLens: Lens-View Visualizations of Evolution Data*, Proceedings 8th International Workshop on Principles of Software Evolution (IWPSE), 2005, pp. 103-112.

[15] O. Maqbool and H.A. Babri, *Hierarchical Clustering for Software Architecture Recovery*, IEEE Transactions on Software Engineering, Vol. 33(11), 2007, pp. 759-780.

[16] J. Kothari, T. Denton, A. Shokoufandeh, S. Mancordis, and A.E. Hassan, *Studying the Evolution of Software Systems Using Change Clusters*, Proceedings 14th IEEE International Conference on Program Comprehension, 2006, pp. 46-55.

[17] R. Robbes, D. Pollet, and M. Lanza, *Logical Coupling Based on Fine-Grained Change Information*, Proceedings 15th Working Conference on Reverse Engineering, 2008, pp. 42-46.

[18] O.I. Lindland, G. Sindre, and A. Sølvberg, *Understanding Quality in Conceptual Modeling*, IEEE Software, 1994, pp. 42-49.

[19] P. van de Laar, P. America, J. Rutgers, S. van Loo, G. Muller, T. Punter, and D. Watts, *The Darwin Project: Evolvability of Software-Intensive Systems*, Third Workshop on Software Evolvability, 2007, pp. 48-53.

[20] http://subversion.tigris.org/

[21] F. Esposito, D. Malerba, V. Tamma, and H.H. Bock, *Classical Resemblance Measures*, Chapter 8.1 of H.H. Bock and E. Diday (Eds.), *Analysis of Symbolic Data: Exploratory Methods for Extracting Statistical Information from Complex Data*, 2000, pp. 139-152.

[22] Baldwin, C. and K. Clark, *Design Rules, Volume 1: The Power of Modularity*, 1999, ISBN 0262024667.