

Software Maintenance Analysis and Understanding of the Software Structure

Jannik Laval, Usman Bhatti, Nicolas Anquetil, and Stéphane Ducasse

RMoD Project-Team
INRIA - Lille Nord Europe
USTL - CNRS UMR 8022, Lille, France
<http://rmod.lille.inria.fr/>

1 Context

1.1 Maintenance Problematic

Most of the effort during a software system lifecycle is spent in supporting its evolution[12]. Maintainers have to deal with large source code systems and have to spend a large part of their time understanding the system. Corbi[3] estimates the portion of time invested in program comprehension to be between 50 and 60%.

Software evolves over time with the modification, addition and removal of new classes, methods, functions, and dependencies. A consequence is that some classes may not be placed in the right packages and the software modularization is broken[5, 6]. As a consequence, software modularization must be maintained. In that respect, it is then important to understand, to assess and to optimize the concrete organization of packages and their relationships.

1.2 Package Granularity

A package is a unit of reuse and deployment: it is built, tested, and released as a whole as soon as one of its classes is changed, or used elsewhere[11]. We name software modularization the organization of classes into multiple packages.

A package provides and requires services from other packages. They can play central roles or peripheral[4]: some packages act as reference hubs, others as authorities. Packages have different usage patterns, often depending on the clients that use them[1]. These multiple views of packages do not ease the understanding and the maintenance.

The remodularization task is to make a better package organization after a structure deterioration. Reengineers have to: (i) understand the structure at package level and at class level and assess its quality; (ii) understand where are structural problems; and (iii) take decisions and verify the impact of these decisions.

Existing approaches lack of (i) a deep understanding fine-grained package structure and dependencies; (ii) an identification of package dependency problems; and (iii) an analyze of the impact of a change on the package structure.

2 Understanding Package Structure

2.1 Identifying cycle causes

We propose eDSM[9], an approach to enrich Dependency Structural Matrix with eCell, a view displaying the internals of a package dependency. We adapt eDSM for software reengineering with contextual information about (i) the type of dependencies (inheritance, class reference, . . .); (ii) the proportion of referencing entities; and (iii) the proportion of referenced entities. We highlight strongly connected component (SCC) and stress potentially simple fixes for cycles using coloring information.

CycleTable[8] presents (i) a heuristic to focus on *shared* dependencies between cycles in Strongly Connected Components; and (ii) a visualization highlighting dependencies to efficiently remove cycles in the system. This visualization is completed with eCell (small views displaying the internals of a dependency).

2.2 Package Layered Structure Identification in presence of Cycles

We propose oZone[7] an approach which provides (i) a strategy to highlight dependencies which break Acyclic Dependency Principle; and (ii) an organization of package in multiple layers even in presence of cycles. While our approach can be run automatically, it also supports human inputs and constraints.

2.3 Supporting Simultaneous Versions for Software Evolution Assessment

We propose Orion[10], an interactive prototyping tool for reengineering to simulate changes and compare their impact on multiple versions of software source code models. Our approach offers an interactive simulation of changes, reuses existing assessment tools, and has the ability to hold multiple and branching versions simultaneously in memory. Specifically, we devise an infrastructure which optimizes memory usage of multiple versions for large models. This infrastructure uses an extension of the FAMIX source code meta-model but it is not limited to source code analysis tools since it can be applied to models in general.

2.4 Metrics analysis

In[2], we study a real structuring case (on the Eclipse platform) to try to better understand if (some) existing metrics would have helped the software engineers in the task. Results show that the cohesion and coupling metrics used in the experiment did not behave as expected and would probably not have helped the maintainers reach there goal.

Currently, we are working on a dependency analysis of packages to measure their relation with their framework. The results show that framework dependencies form a considerable portion of the overall package dependencies. This means that low cohesive packages should not considered systematically as package of low quality.

References

1. H. Abdeen, I. Alloui, S. Ducasse, D. Pollet, and M. Suen. Package reference fingerprint: a rich and compact visualization to understand package relationships. In *European Conference on Software Maintenance and Reengineering (CSMR)*, pages 213–222. IEEE Computer Society Press, 2008.
2. N. Anquetil and J. Laval. Legacy software restructuring: Analyzing a concrete case. In *CSMR 2011: Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, Oldenburg, Germany, 2011.
3. T. A. Corbi. Program understanding: Challenge for the 1990’s. *IBM Systems Journal*, 28(2):294–306, 1989.
4. S. Ducasse, D. Pollet, M. Suen, H. Abdeen, and I. Alloui. Package surface blueprints: Visually supporting the understanding of package relationships. In *ICSM ’07: Proceedings of the IEEE International Conference on Software Maintenance*, pages 94–103, 2007.
5. S. Eick, T. Graves, A. Karr, J. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12, 2001.
6. W. G. Griswold and D. Notkin. Automated assistance for program restructuring. *ACM Trans. Softw. Eng. Methodol.*, 2(3):228–269, 1993.
7. J. Laval, N. Anquetil, and S. Ducasse. Ozone: Package layered structure identification in presence of cycles. In *Proceedings of the 9th edition of the Workshop BELgian-NEtherlands software eVOLution seminar, BENEVOL 2010*, Lille, France, 2010.
8. J. Laval, S. Denier, and S. Ducasse. Cycles assessment with cycletable. Technical report, INRIA, 2010.
9. J. Laval, S. Denier, S. Ducasse, and A. Bergel. Identifying cycle causes with enriched dependency structural matrix. In *WCRE ’09: Proceedings of the 2009 16th Working Conference on Reverse Engineering*, Lille, France, 2009.
10. J. Laval, S. Denier, S. Ducasse, and J.-R. Falleri. Supporting simultaneous versions for software evolution assessment. *Journal of Science of Computer Programming (SCP)*, May 2010.
11. R. C. Martin. Design principles and design patterns, 2000. www.objectmentor.com.
12. I. Sommerville. *Software Engineering*. Addison Wesley, fifth edition, 1996.