Reverse Engineering through Holistic Software Exploration

Mircea Lungu *and* Michele Lanza
Faculty of Informatics - University of Lugano, Switzerland

Abstract

Software Exploration tools usually work at a single level of abstraction. We argue for an approach which integrates multiple levels of abstraction in exploration. Each of these levels presents complementary information which is useful for the reverse engineering and understanding process.

Introduction. Software exploration is a technique to analyze software projects. Most software exploration tools work at a single level of abstraction. We propose the concept of *holistic software exploration* which encourages exploring multiple levels of abstraction at the same time.

We distinguish the following abstraction levels that contain useful information about software projects:

- The super-project level, i.e., the context or contexts in which a project exists. We are not aware of research in exploration at this level.
- The project macro level, i.e., the modules composing a system and the interactions between them
- The project micro level, i.e., fine-grained entities such as classes and methods

Super-Project Level. Software projects are *social animals*. They do not exist alone but in the context of other projects. Companies and research groups use versioning repositories in which their projects cohabitate. In the extreme case, super-repositories such as SourceForge can also be considered as the ultimate context of the many open-source projects hosted on it. Sometimes the system's eco-system, i.e., its environment, can provide interesting information about the project.

The screenshot Figure 2 presents one application to perform exploration at super-project level. The application is *The Small Project Observatory* [1]. The conventions are:

- The projects have different colors and the graphics are stacked one on top of the other.
- The time is divided in equal intervals (*e.g.,* months)

1 See http://www.inf.unisi.ch/lungu/observatory

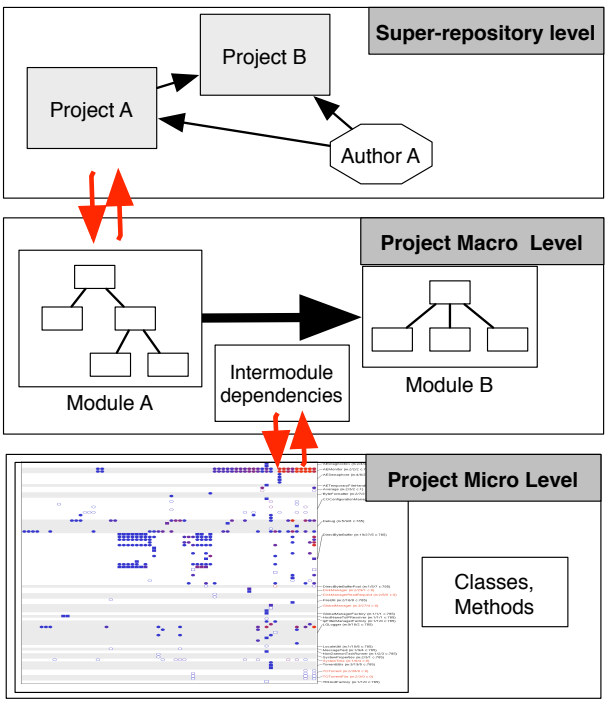

Figure 1. The Concept of Vertical Software Exploration

- The height of the intervals reflects the size and respectively activity of the projects in terms of classes and commits per month
- If the projects are not changed during an interval the color of the interval is 50% transparent

If one looks only at each project as an individual (see Figure 2), he can observe that after the middle of 2005 the size of the bottom project did not change. However, if one looks at information in the whole context of the project, one can see that there is a correlation between the time the bottom project stops and the top project starts to grow. Another observation is that the activity of the blue project stops and the activity of the brown one starts. A new repository was started, with the contents of the old project, while the old

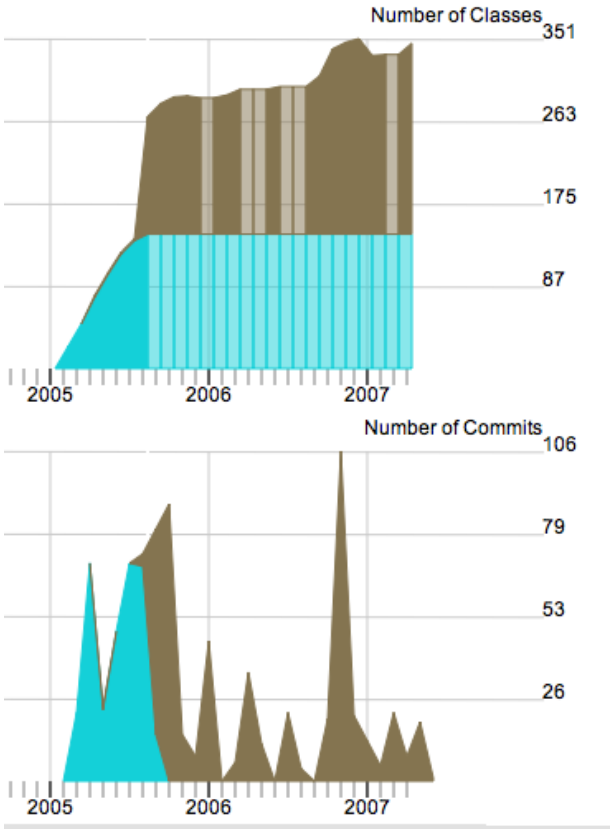

Figure 2. The evolution of size and activity for all the projects in the Bern Store which have 'lungu' as author.

one remained in place.

We extract other types of information, such as author collaborations, inter-project code sharing, information about inter-project dependencies, etc.

Project Macro Level. The super-repository level is useful up to a point. After that, the reverse engineer has to focus the analysis at a lower abstraction level. We consider the next level to be the one where modules and their interdependencies are presented. Softwarenaut [4, 3, 2] (see Figure 3), our module interdependency exploration tool is built on top of Moose[1].

Usually at this level the projects are decomposed hierarchically and one can explore the hierarchy.

Project Micro Level. The next level of abstraction is the level of classes and methods. There are various ways in which one can arrive to this level coming from the previous one. One is to explore a certain module in detail, disconnecting it from the rest of the system. Another one is to explore a relationship between two modules, such as we

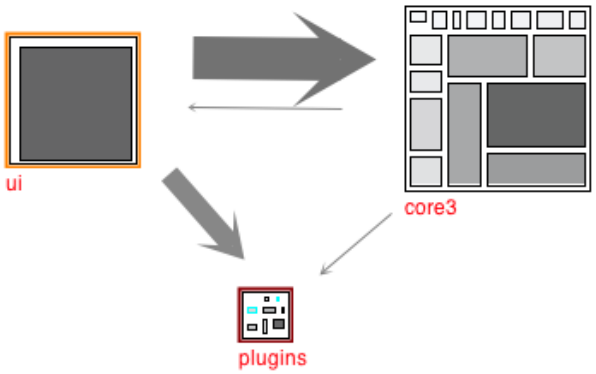

Figure 3. The dependencies between three top-level modules in Azureus

did in our work on intra-dependency analysis [3] (see Figure 4).

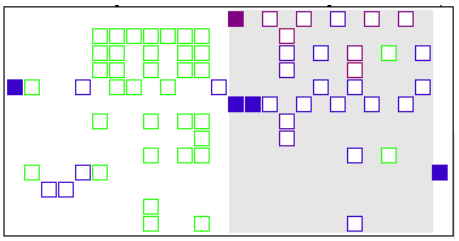

Figure 4. The Semantic Dependency Matrix is an intra-dependency visualization technique

References

[1] S. Ducasse, T. Gîrba, M. Lanza, and S. Demeyer. Moose: a collaborative and extensible reengineering environment. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 55–71. Franco Angeli, Milano, 2005.

[2] M. Lungu, A. Kuhn, T. Gîrba, and M. Lanza. Interactive exploration of semantic clusters. In *3rd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2005)*, pages 95–100, 2005.

[3] M. Lungu and M. Lanza. Softwarenaut: Cutting edge visualization. In *Proceedings of Softvis 2006 (3rd International ACM Symposium on Software Visualization)*, pages 179–180. ACM Press, 2006.

[4] M. Lungu and M. Lanza. Softwarenaut: Exploring hierarchical system decompositions. In *Proceedings of CSMR 2006*, pages 349–350, Los Alamitos CA, 2006. IEEE Press.