or smaller elements (like number of characters of a method definition), while different coupling metrics were used which represent different intensities of mutual uses of system components. The measures and links between the hyperdocuments which our tool has to generate are configurable. They have the form of parameterised SQL-statements in an HTML-skeleton file. The SQL statements are used to extract the relevant information from a database which represents the symbol-table information of a CASE-Tool (cf. [LeSi98]).

We developed the thesis that the understandability of a large software system can be improved by adding this type of explicit documentation to the system. A small exploratory student experiment with ten participants was performed which was used to validate our approach; in this experiment, both our tool and other approaches were used to restructure given software systems into subsystems, i.e. the systems were given as flat sets of files which were to be grouped into several subsystems (cf. [AbPeSo98]). To be able to compare the results with respect to the effort spent, we restricted the time for each restructuring to 120 minutes. This was our way to operationalise understandability. Each student restructured two systems (each with about 60 classes), one with the information presented by our tool and one without, in which source-code analysis was performed in any other way the students chose to. We collected information about which kind of information the students used in the projects, and we compared the different results, e.g. with respect to forgotten classes. Our basic assumptions were confirmed, e.g. when our tool was used, the students used more different types of information in the same time than when it was not used. Students reported that the extra data presented by our tool was helpful for their understanding.

References

[AbPeSo98] F. B. e Abreu, C. Pereira, P. Sousa. "Reengineering the Modularity of Object Oriented Systems", in Workshop "Techniques, Tools and Formalisms for Capturing and Assessing the Architectural Quality in Object Oriented Software", ECOOP'98.

[LeSi98] C. Lewerentz, F. Simon. "A Product Metrics Tool Integrated into a Software Development Environment", in Proceedings of Workshop on Object-Oriented Product Metrics for Software Quality Assessment, ECOOP'98, CRIM Montréal 1998.

[KöRuSi98] G. Köhler, H. Rust, F. Simon. "An Assessment of Large Object Oriented Software Systems", in Proceedings of Workshop on Object-Oriented Product Metrics for Software Quality Assessment, ECOOP'98, CRIM Montréal 1998.

Reverse Engineering Based on Metrics and Program Visualization

Authors: Michele Lanza, Stéphane Ducasse and Serge Demeyer
Emails: {`lanza,ducasse,demeyer`}`@iam.unibe.ch`
URLs: `http://www.iam.unibe.ch/~{lanza,ducasse,demeyer}/`

The reverse engineering of large scale object-oriented legacy systems is a challenging task with a definite need for approaches providing a fast overview and focussing on the problematic parts. We investigate a hybrid approach, combining the immediate appeal of visualizations with the scalability of metrics. Moreover, we impose ourselves the extra constraint of simplicity: i.e. (a) that the graph layout should be *quite trivial* and (b) that the extracted metrics should be *simple* to compute. Indeed, our goal is to identify useful combinations of graphs and metrics that can be easily reproduceable by reverse engineers using some scriptable reengineering toolset. We validate such a hybrid approach by showing how CodeCrawler —the experimental platform we built— allowed us to reverse engineer a small to medium software system.

Principle. We enrich a simple graph with metric information of the object-oriented entities it represents. In a two-dimensional graph we render up to five metrics on a single node at the same time.

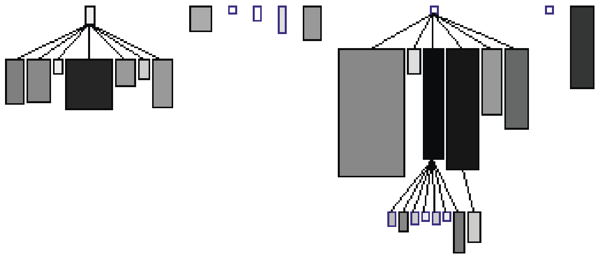

Fig. 1. Inheritance Tree; node width = NIV, node height = NOM and color = NCV.

As an example 1 shows an inheritance tree graph of CodeCrawler. The nodes represent the classes, the edges represent the inheritance relationships. The size of the nodes reflects the number of instance variables (width) and the number of methods (height) of the class, while the color tone represent the number of class variables. The position of a node does not reveal a metric as it is used to show the location in the inheritance tree.

CodeCrawler. CodeCrawler is developed within the VisualWorks 2.0 Smalltalk environment, relying on the HotDraw framework [John92] for its visualization. Moreover, it uses the facilities provided by the VisualWorks 2.0 environment for the Smalltalk code parsing, whereas for other languages like C++ and Java it relies on Sniff+ to generate code representation coded using the FAMIX Model [Tich98]. For more information see
`http://www.iam.unibe.ch/~lanza/codecrawler/`

References

[**Duca99**] S. Ducasse, S. Demeyer and M. Lanza, A Hybrid Reverse Engineering Approach Combining Metrics and Program Visualization, Accepted to WCRE'99.
[**Lanz99**] M. Lanza, Master thesis, Combining Metrics and Graphs to Reverse Engineer OO Applications, University of Berne, 1999.