CodeCrawler - A Lightweight Software Visualization Tool

Michele Lanza (lanza@iam.unibe.ch)
Software Composition Group - University of Bern, Switzerland

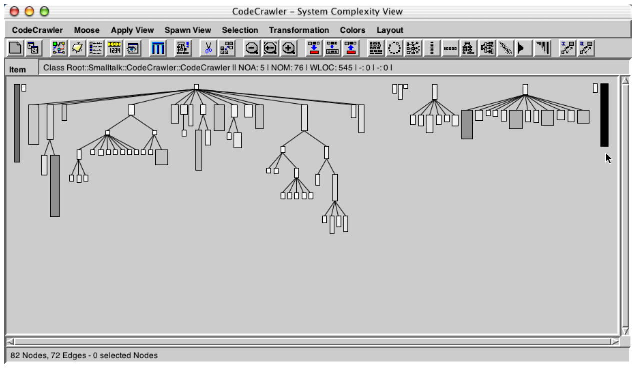
Abstract

CodeCrawler is a language independent software visualization tool. It is mainly targeted at visualizing object-oriented software, and in its newest implementation it has become a general information visualization tool. It has been validated in several industrial case studies over the past few years. It strongly adheres to lightweight principles: CodeCrawler implements and visualizes polymetric views, *lightweight visualizations of software enriched with semantic information such as software metrics and source code information.*

1 Introduction

CodeCrawler is a lightweight software visualization tool, whose first implementation dates back to 1998 and it has been implemented as part of Lanza's Ph.D. thesis [3]. In the meantime it has been evolved into an information visualization framework, and has been customized to work in contexts like website reengineering and concept analysis. It keeps however a strong focus on software visualization. CodeCrawler is a language independent SV tool, because it uses the Moose reengineering environment [2] which implements the FAMIX metamodel [1], which among other languages models software written in C++, Java, Smalltalk, Ada, Python, COBOL, etc.

In Figure 1 we see CodeCrawler visualizing itself with a polymetric view called *System Complexity*. The metrics used in this view are the number of attributes for the width, the number of methods for the height, and the number of lines of code for the color of the displayed class nodes.



Class Root::Smalltalk::CodeCrawler::CodeCrawler || NOA: 5 | NOM: 76 | WLOC: 545 | -: 0 | -: 0 |

82 Nodes, 72 Edges - 0 selected Nodes

Figure 1. A screenshot of CodeCrawler visualizing itself with a *System Complexity* **view.**

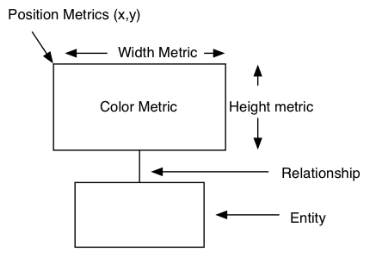


Figure 2. The principle of a polymetric view.

2 The Principle of a Polymetric View

In Figure 2 we see that, given two-dimensional nodes representing entities and edges representing relationships, we enrich these simple visualizations with up to 5 metrics on these node characteristics:
Node Size. The width and height of a node can render two measurements. We follow the convention that the wider and the higher the node, the bigger the measurements its size is reflecting.
Node Color. The color interval between white and black can display a measurement. Here the convention is that the higher the measurement the darker the node is. Thus light gray represents a smaller metric measurement than dark gray.
Node Position. The X and Y coordinates of the position of a node can reflect two other measurements. This requires the presence of an absolute origin within a fixed coordinate system, therefore not all layouts can exploit this dimension.

3 Example Polymetric Views

CodeCrawler visualizes three different types of polymetric views:

Coarse-grained views. Such views are targeted at visualizing very large systems (*e.g.*, over 100 kLOC to several MLOC). In Figure 3 we see a *System Hotspots* view of 1.2 million lines of C++ code. The view uses the number of methods for the width and height of the class nodes. We gather for example from this view that there are classes with several hundreds of methods (at the bottom).

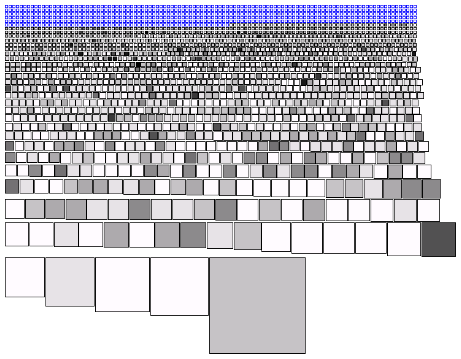


Figure 3. A *System Hotspots* view on 1.2 MLOC of C++ code.

Fine-grained views. The most prominent view is the *Class Blueprint* view, a visualization of the internal structure of classes and class hierarchies [4]. In Figure 4 we see a class blueprint view of a small hierarchy of 4 classes. The class blueprint view helped to develop a pattern language [3]. In the present example we see the patterns pure overrider, siamese twin, template method design pattern, and template class. The limited size of this paper does not allow us to deepen this discussion, please refer to [3] for more details.

Evolutionary views. The most prominent view is the *evolution matrix* view, a visualization of the evolution of complete software systems [5]. In Figure 5 we see an example of such a visualization, which again allows us to develop a pattern language applicable in the context of software evolution.

4 Features of CodeCrawler

Moreover, CodeCrawler features grouping support, customizable views, has been industrially validated, and is being used if software industry mainly by consultants. CodeCrawler is freeware and can be obtained at http://www.iam.unibe.ch/∼lanza/

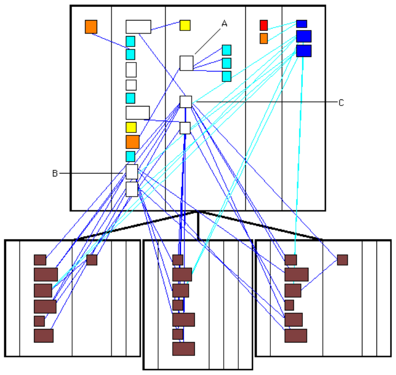


Figure 4. A *Class Blueprint* view on a small hierarchy of 4 classes written in Smalltalk.

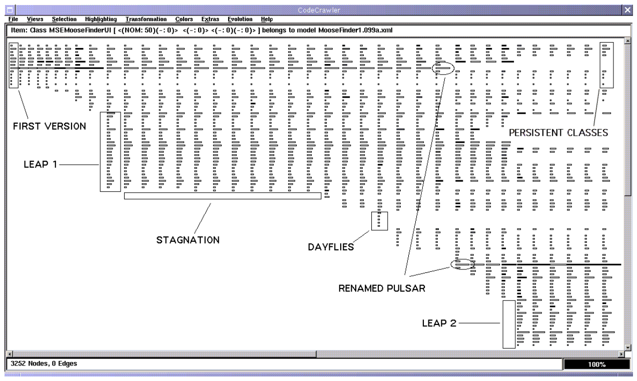


Figure 5. An *Evolution Matrix* view on 38 versions of an application written in Smalltalk.

References

[1] S. Demeyer, S. Tichelaar, and S. Ducasse. FAMIX 2.1 – the FAMOOS information exchange model. Technical report, University of Bern, 2001.

[2] S. Ducasse, M. Lanza, and S. Tichelaar. Moose: an extensible language-independent environment for reengineering object-oriented systems. In *Proceedings of the Second International Symposium on Constructing Software Engineering Tools (CoSET 2000)*, June 2000.

[3] M. Lanza. *Object-Oriented Reverse Engineering - Coarse-grained, Fine-grained, and Evolutionary Software Visualization*. PhD thesis, University of Berne, may 2003.

[4] M. Lanza and S. Ducasse. A categorization of classes based on the visualization of their internal structure: the class blueprint. In *Proceedings of OOPSLA 2001*, pages 300–311, 2001.

[5] M. Lanza and S. Ducasse. Understanding software evolution using a combination of software visualization and software metrics. In *Proceedings of LMO 2002*, pages 135–149, 2002.