CCJun - Polymetric Views in Three-dimensional Space

Student Project

Author

Christoph Wysseier

May 2004

Supervised by:

Dr. Michele Lanza
Prof. Dr. Oscar Nierstrasz

Institut für Informatik und angewandte Mathematik
Universität Bern

The address of the author:

Christoph Wysseier
Gesellschaftsstr. 38
CH-3012 Bern
chris@netstyle.ch

Abstract

CodeCrawler is a tool to visualize software systems using polymetric views as a lightweight reverse engineering approach. In this project we introduce the extension CCJun, which enriches the polymetric views with a third dimension and shows how to generalize the visualization engine of CodeCrawler.

Contents

Chapter 1

Introduction

CodeCrawler is a tool which implements polymetric views as a lightweight reverse engineering approach based on simple software visualisations enriched with software metrics [LANZ 03a]. For this we chose Jun [1] as 3D framework to realize three-dimensional graphics. With CCJun we implemented a bridge between CodeCrawler and Jun to realize three-dimensional polymetric views.

1.1 Software Visualisation and Metrics

Software visualization is widely used to infer information about software systems. Lanza's approach of polymetric views which are enriched with up to five software metrics is the basic concept of CodeCrawler. With software metrics there exists an instrument to measure the quality and complexity of software systems as a source of information for software reengineering. Metrics measure properties of a software system by mapping them to numbers.
The graphical represenation of a software system is realized by displaying nodes for source code artifacts (*e.g.* classes, methods, attributes, etc.) and edges for relationships (*e.g.* inheritance, invocation, etc.). The visual representation of software systems is then extended with additional information computed by software metrics. The results are added to the view by manipulating the size, position and color of a node or by manipulating the width and color of an edge. The result is a two-dimensional view called polymetric view of a software system.

1.2 CodeCrawler

CodeCrawler is a tool that supports the representation of polymetric views [LANZ 03b, LANZ 03c]. CodeCrawler uses as visualization engine a package called HotDraw [BRAN 95, JOHN 92] which is a lightweight 2D editor written in Smalltalk.
The goal of this project was to extend the graphical capabilities of CodeCrawler

[1] http://www.sra.co.jp/people/aoki/Jun/

up to the third dimension. In order to do so we had to generalize the visualization engine of CodeCrawler. Therefore we developed the package CCJun which works as a bridge between CodeCrawler and Jun, a 3D graphics library for Smalltalk.

1.3 Structure of this document

In Chapter 2 we describe the internal architecture and the visualization engine of CodeCrawler. Furthermore the implementation of CCJun is described in this chapter.
Chapter 3 shows some applications of CCJun.
The last chapter (Chapter 4) introduces some future work.

Chapter 2

Description

In this chapter we introduce the approach of polymetric views. Then we describe the overall architecture of CodeCrawler with a focus on the visualization engine. We then show the changes that have been made to CodeCrawler's internal architecture and the implementation of CCJun as an extension to display three-dimensional polymetric views.

2.1 Polymetric Views

In Figure 2.1 we see that, given two-dimensional nodes representing entities and edges representing relationships, we enrich these simple visualizations with up to 5 metrics:
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.
Edge Color. The only metric applicated to edges is their color.

2.2 CodeCrawlers Overall Architecture

CodeCrawler adopts what we call a bridge architecture, as we see in Figure 2.2: the internal architecture, e.g., the core of CodeCrawler, acts as a bridge between the visualization engine (on the left) and the metamodel (on the right). It uses as visualization engine the HotDraw framework and as metamodel the FAMIX metamodel [DEME 01], whose implementation is called the Moose reengineering environment [DUCA 00] [DUCA 01].

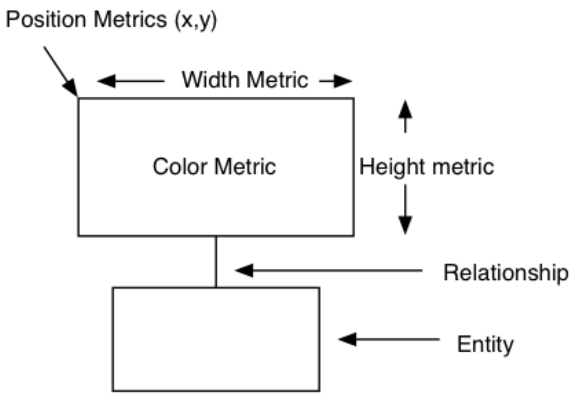


Figure 2.1: The principle of polymetric views

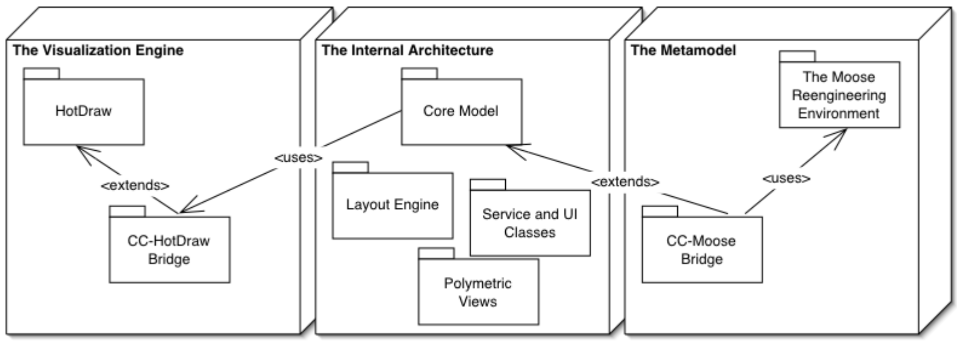


Figure 2.2: The general architecture of CodeCrawler, composed of three main subsystems: the core, the metamodel and the visualization engine.

2.3 CodeCrawlers Visualization Engine

CodeCrawler is tightly coupled to the 2D graphic editor HotDraw which provides all the basic functionality used to display polymetric views. Polymetric views consist mainly of nodes and edges which are mapped for the graphical representation to rectangle and line figures.
Therefore Codecrawler uses two small hierarchies which are representing the figures on one hand and figure models on the other hand. This differentiation is necessary to be protected against changes in HotDraw (See Figure 2.3).

2.3.1 Figures

CodeCrawler offers the classes *CCRectangleFigure* and *CCLineFigure* to represent the nodes and edges on the screen. These classes are subclassing the simple Rectangle and Line classes of HotDraw and are adding the additional functionality needed by CodeCrawler. In fact there exist more figures such as *CCNamedFigure*

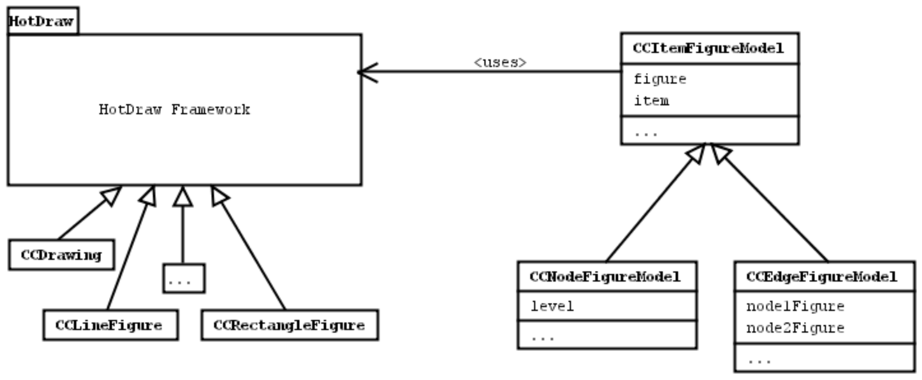


Figure 2.3: This simple UML diagram shows how CodeCrawler extends the HotDraw framework.

and others but those are only special representations of the already presented ones. The drawing of the figures on the canvas is realized using the class *CCDrawing*.

2.3.2 Figure Models

To be protected against changes of HotDraw there exists another small hierarchy which serves as bridge between CodeCrawlers MetaModel and the HotDraw Figures. The three classes *CCItemFigureModel*, *CCNodeFigureModel* and *CCEdgeFigureModel* are implementing figure operation (graphical operations, geometric transformations, etc.).

2.4 Jun - A 3D Graphics Framework

Jun is a large 3D graphics framework with support for OpenGL[1]. Besides other features it provides a hierarchy that allows to create OpenGL objects which are displayed on screen using its own user interface. OpenGL is a good choice because it is available on a lot of platforms incorporated in the operating system.

2.5 Implementation of CCJun

We implemented the tool CCJun to be able to create 3D views of polymetric views. As the visualization engine we chose Jun. Because the goal was a platform-independent implementation we chose the OpenGL objects of Jun to implement the new functionality.

[1]http://www.opengl.org/

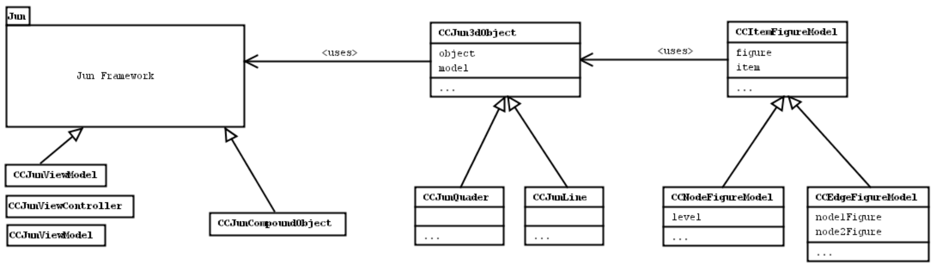


Figure 2.4: Modified visualization engine to work with the 3D graphics framework Jun.

2.5.1 Figures

To realize polymetric views without changing their simplicity we created the 3D figures according to the implementation of the HotDraw figures. The only difference is the added third dimension which led us from rectangles to quaders and from lines to cylinders.
Because of the different architecture of Jun these figures are implemented using a single class called *CCJunCompoundObject* which inherits from *JunOpenGLCompoundObject*. This class offers the creation of a lot of different figures which are plugged together using polygons.

2.5.2 Bridge to CodeCrawler

Our goal was to provide the same functionality as for the HotDraw figures and mapping them directly to Jun figures. This functionality includes positioning and coloring the figures and even user interaction.
Our implementation oriented itself at the architecture of the figure models. Therefore we built a little hierarchy using the classes *CCJun3dObject*, *CCJunLine* and *CCJunQuader*. We can see a simple class diagram in Figure 2.4 which shows also the connection to the figure models of CodeCrawler.
The root of this hierarchy is used to provide similar behavior for each 3D object. This includes geometrical transformations, coloring and selection behavior.
The figure *CCJunQuader* builds its own 3D figure by creating a box with the specified height, width, depth and color. Because there is today no additional metric for the height we chose a standard value for it. The figure *CCJunLine* builds a cylinder as 3D figure which is then connected from one node figure to another.

2.5.3 Building polymetric views in 3D

With the new figures and the bridge to CodeCrawler we are now able to create polymetric views in three-dimensional space. In CodeCrawler the polymetric views are arranged by the view specification manager called *CCViewSpecManager*. Every view knows among other things its figure class which is used to build the view. To create 3D views we extended the default attitude in *CCViewSpecManager* to use the new implemented figures.

2.5.4 User Interface of CCJun

Once the view is rendered on screen the user also wants interact with it. Therefore CodeCrawler provides context-based menus which are adapted to the currently selected node(s). The goal of CCJun was to develop a user interface that includes similar features.
The user interface of CCJun is an extended version of viewfinder (see Figure 2.5), the integrated user interface of Jun. It provides an easy-to-use navigation interface with zooming, rotating and selecting. To realize the context-based pop-up menus on nodes or edges we extended this interface so that on selecting a node or edge the menu is displayed. The menus are rendered depending on the type of object that is selected.
The user interface is implemented using the Model-View-Controller pattern. CCJun implements a controller (*CCJunDisplayController*) to provide mouse interacting facilities, a model (*CCJunDisplayModel*) and a view (*CCJunDisplayView*). These classes are shown in Figure 2.4.
An interesting feature is the possible parellel use of CodeCrawlers two-dimensional views and CCJun. It is for instance possible to select a few nodes in the 2D view and to display them in 3D using CCJun. Like this a system reenginerer may decide which part of a view should be visualized in 3D.
Using the user interface of CodeCrawler it is even possible to choose different layouts which are rendered directly in CCJun. Like this it is possible to change the layout to get a different view of a software system.

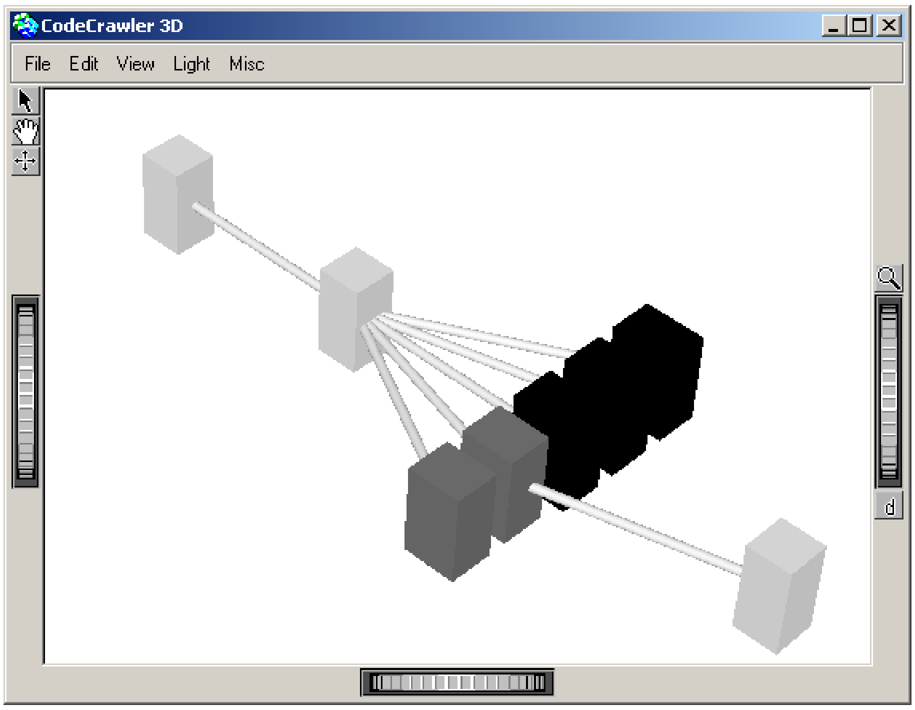


Figure 2.5: A little hierarchy displayed in the user interface of CCJun.

Chapter 3

Exemplification

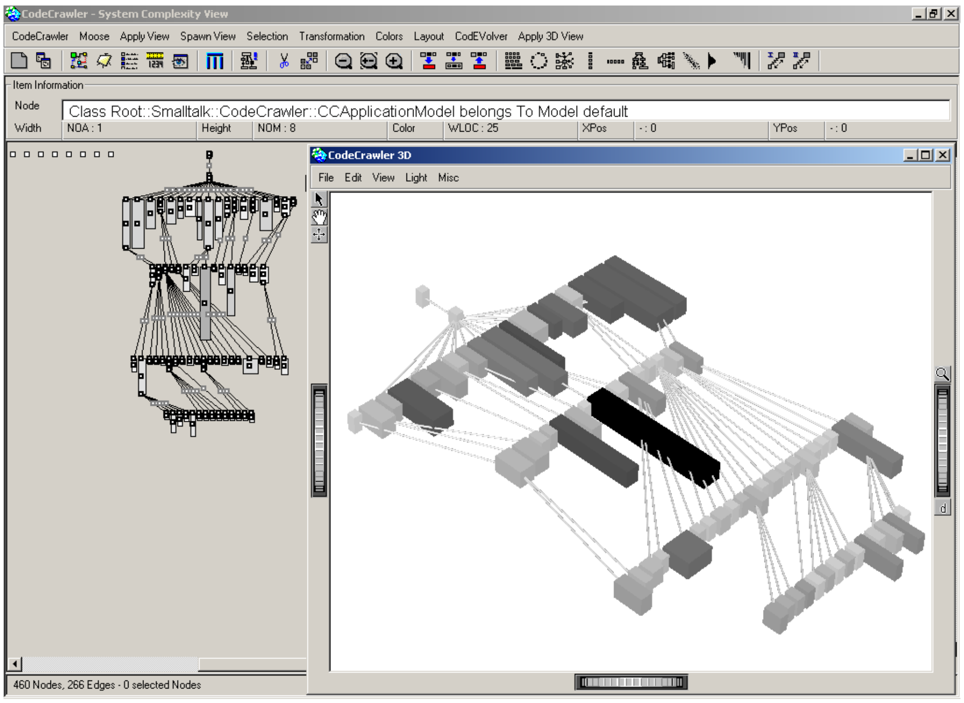
In this chapter we present some example screenshots realized using CCJun. The examples are focused to the interaction possibilities with CodeCrawler.

3.1 Creating three-dimensinal Views

An important feature of CCJun was its integration into CodeCrawler's user interface. Like this it is possible to use the given selection menus to select nodes or edges to be displayed in CCJun's user interface.
In Figure 3.1 you can see the internal architecture of CodeCrawler displayed using CCJun. The items were selected directly from the two-dimensional view of the complete representation of CodeCrawler's architecture. As you can see the view is similar but CCJun offers new interaction possibilities like rotating the whole hierarchy.

3.2 The User Interface of CCJun

The goal of this project was to implement a three-dimensional representation of polymetric views without losing the interaction possibilities of CodeCrawler. One interaction possibility is shown in Figure 3.2. The user is able to select an item directly from the three-dimensional view. Using the right mouse button the context-based menu to the selected item is shown. The hierarchy displayed is the same as in Figure 3.1.
In this example the menu provides the possibilities to inspect the figure, spawn a two-dimensional blueprint of the class and to browse through the code. To better recognize the selected node it is marked using a default color (*i.e.* green).


CodeCrawler - System Complexity View
CodeCrawler Moose Apply View Spawn View Selection Transformation Colors Layout CodEVolver Apply 3D View
CodeCrawler 3D
File Edit View Light Misc

Item Information

Node: Class Root::Smalltalk::CodeCrawler::CCApplicationModel belongs To Model default

Width: NOA : 1 | Height: NOM : 8 | Color: WLOC : 25 | XPos: - : 0 | YPos: - : 0

460 Nodes, 266 Edges - 0 selected Nodes

Figure 3.1: In this figure the nodes to display in CCJun are chosen using Code-Crawler's two-dimensional view.

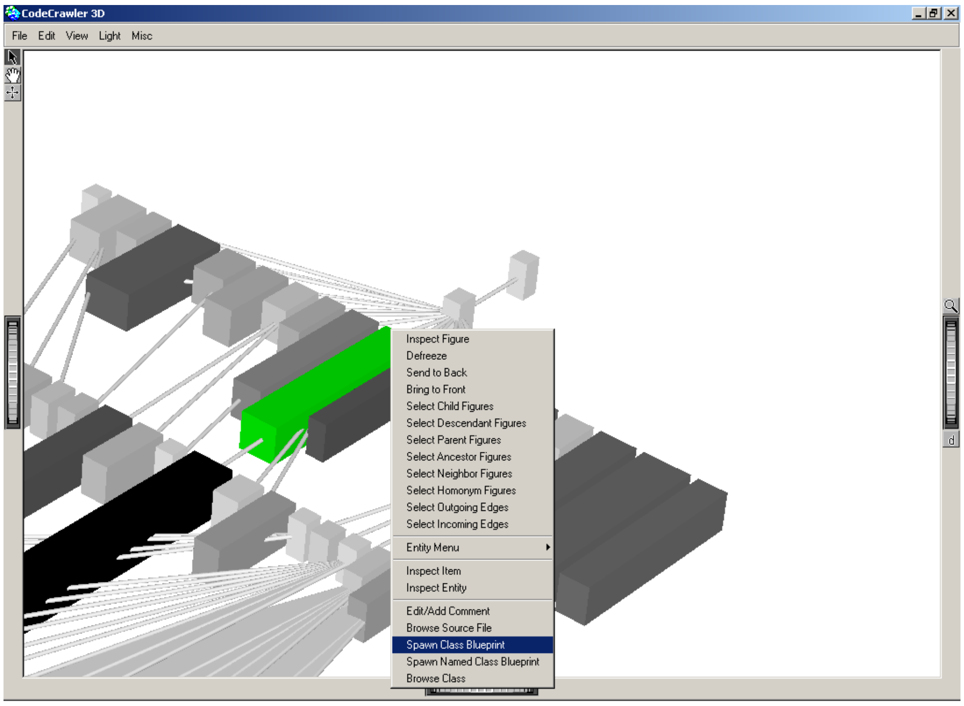


Figure 3.2: This figure shows the context-based menu on a node.

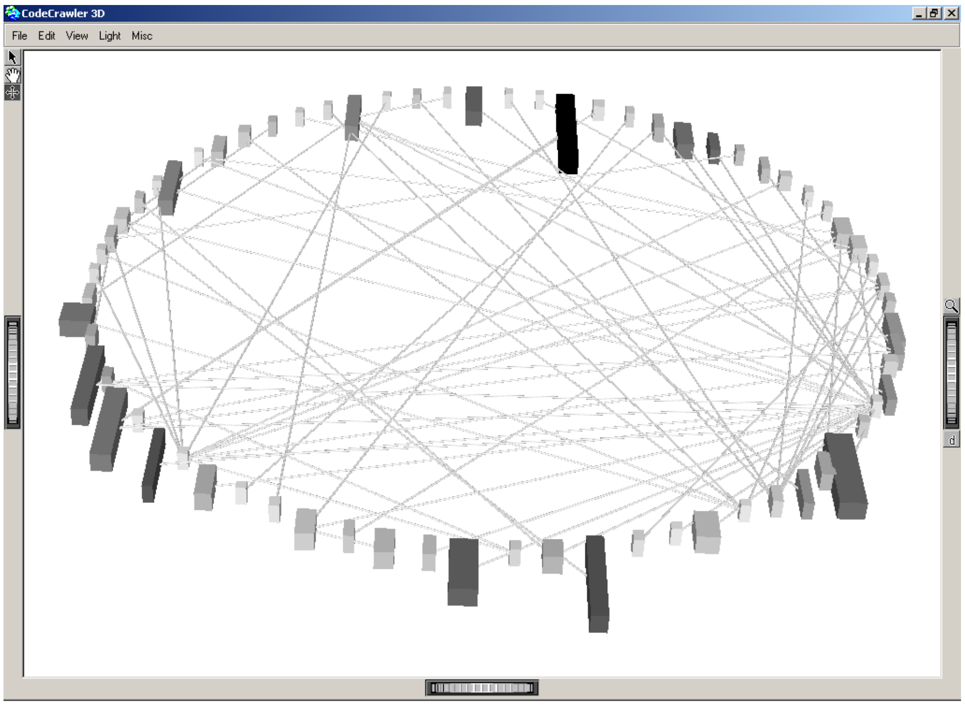


Figure 3.3: This figure was realized using the Circle Layout chosen from CodeCrawler's user interface.

3.3 Different Layouts

The same hierarchy displayed using another layout is shown in Figure 3.3. Using the Circle Layout the nodes are placed on a circle while the edges still connect the same edges. Changing the layout of a view may be executed without choosing the nodes again.

Chapter 4

Future Work

In this chapter we present some possible future work.

- **Three-dimensional polymetric views and semantics:** One of the first steps to be done exploring the possibilities of three-dimensional views is to add another metric to the polymetric views. To not lose the simplicity of this approach one must have an eye on the semantics.
- **Adding extra information to three-dimensional views:** Using the OpenGL framework of Jun it is also possible to use other forms of visual information representation. Possible ideas would be the use of textures or transparency.
- **Representing runtime information** An interesting research topic would be to visualize runtime information as an animation of three-dimensional views. This approach would provide another form of exploring a software system during runtime which would be helpful to a software reengineer.

List of Figures

Bibliography

[BRAN 95] J. Brant. HotDraw. Master's thesis, University of Illinois at Urbana-Chanpaign, 1995. (p 1)

[DEME 01] S. Demeyer, S. Tichelaar, and S. Ducasse. *FAMIX 2.1 — The FAMOOS Information Exchange Model*. Research report, University of Bern, 2001. (p 3)

[DUCA 00] 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. (p 3)

[DUCA 01] S. Ducasse, M. Lanza, and S. Tichelaar. *The Moose Reengineering Environment*. Smalltalk Chronicles, August 2001. (p 3)

[JOHN 92] R. E. Johnson. *Documenting Frameworks using Patterns*. In Proceedings OOPSLA '92, volume 27, pages 63–76, October 1992. (p 1)

[LANZ 03a] M. Lanza. *Object-Oriented Reverse Engineering — Coarse-grained, Fine-grained, and Evolutionary Software Visualization*. PhD thesis, University of Berne, Mai 2003. (p 1)

[LANZ 03b] M. Lanza. *CodeCrawler — Lessons Learned in Building a Software Visualization Tool*. In Proceedings of CSMR 2003, pages 409–418. IEEE Press, 2003. (p 1)

[LANZ 03c] M. Lanza and S. Ducasse. *Polymetric Views — A Lightweight Visual Approach to Reverse Engineering*. IEEE Transactions on Software Engineering, vol. 29, no. 9, pages 782–795, September 2003. (p 1)