

Università  
della  
Svizzera  
italiana

Faculty  
of Informatics

---

# Lateralus

Visual analysis of SVN commit history

---

**Christian Caggiano**

**Supervised in 2009 by**  
Prof. Michele Lanza



# Abstract

The purpose of this project is to implement an appealing and information bearing visualization feedback of the lifetime of a software system, based on addition, deletion and modification of the files composing the mentioned system. In particular to visualize the relevant modification of a software project, and changes that occur during collaborative development.

This tool is intended to give a visual interpretation to modifications, related to the development of a project, and monitor the evolution of a software system through time, based on the log files of the SVN versioning system.

Given the definition of software visualization: Software Visualization is concerned with the static or animated 2-D or 3-D visual representation of information about software systems based on their structure, history, or behavior. Software Visualization supports the understanding of software systems and algorithms and helps identifying their anomalies.

The proposed solution is a visual software evolution analysis tool, developed with Processing 1.0+ programming language, that allows to visualize relevant information and characteristics about the lifetime of a system, in particular:

Support the understanding of the software system Analyze the system development to track anomalies Maximize the amount of relevant information about the system

# Acknowledgments

I want to thank professor Lanza for his patience, support and for the precious advise and my family for their unconditional support, love and patience. Also I'd like to thank all my friends and peers at the faculty, Jacopo, Marco, Daniele, Alessandro, Andi, Alessio, Gilli and Paolo, and part of the academic staff of the faculty, in particular Marco D'Ambros and Richard Wettel.

# Contents

<b>Acknowledgments</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Perception and Cognition . . . . .	1
1.2 Analytical Reasoning . . . . .	2
1.3 Three levels of Brain Vision . . . . .	2
1.4 Preattentive Processing . . . . .	3
1.4.1 Preattentive Attributes . . . . .	3
<b>2 Software Visualization</b>	<b>5</b>
2.1 Historical overview . . . . .	5
2.1.1 Petroglyphs . . . . .	5
2.1.2 Maps and Historical Imagery . . . . .	6
2.1.3 Modern Scientific Visualization . . . . .	6
2.2 Applied Software Visualization . . . . .	7
2.3 Software Evolution Visualization . . . . .	9
<b>3 Related Work</b>	<b>11</b>
3.1 Software Visualization Tools . . . . .	11
3.1.1 CodeCrawler . . . . .	11
3.1.2 X-Ray . . . . .	12
3.2 Software Evolution Visualization Tools . . . . .	13
3.2.1 CodeCity . . . . .	13
3.2.2 Chronia . . . . .	13
3.2.3 BugCrawler . . . . .	14
3.2.4 Code Swarm . . . . .	15
<b>4 Lateralus</b>	<b>16</b>
4.1 Idea . . . . .	16
4.2 Overview . . . . .	16
4.3 Features . . . . .	17
4.4 Visual Metaphors . . . . .	17
4.4.1 Abstraction and Rendering . . . . .	17
4.4.2 Visualizing Files: Squares and Cubes . . . . .	18
4.4.3 Visualizing Age: Layout and Position . . . . .	18
4.4.4 Visualizing Changes: Heat map . . . . .	19
4.5 Views . . . . .	21
4.5.1 2D View . . . . .	21
4.5.2 3D View . . . . .	21
4.5.3 Experimental 3D View . . . . .	21
4.6 Implementation . . . . .	23
4.6.1 Behavioral View . . . . .	23

---

4.6.2	Structural View . . . . .	24
4.7	Components . . . . .	24
4.7.1	Subversion SVN . . . . .	24
4.7.2	Processing . . . . .	24
4.7.3	External libraries . . . . .	25
<b>5</b>	<b>Lateralus Validation</b>	<b>26</b>
5.1	GNOME Applications . . . . .	26
5.1.1	Test Case 1: Nautilus . . . . .	26
5.1.2	Test Case 2: Epiphany . . . . .	30
5.1.3	Test Case 3: Totem . . . . .	36
<b>6</b>	<b>Conclusion</b>	<b>41</b>
6.1	Future Work . . . . .	41

# List of Figures

1.1	<b>Attentive Processing</b>	An example of attentive processing . . . . .	3
1.2	<b>Preattentive Processing</b>	An example of preattentive processing . . . . .	3
1.3	<b>Graphical primitives</b>	Example of preattentive attributes. . . . .	4
2.1	<b>Petroglyphs</b>	Rock paintings from 30'000 years ago. . . . .	6
2.2	<b>Historical Atlas of Europe</b>	Historical map from the 16th century . . . . .	7
2.3	<b>Example of Scientific Visualization</b>	3D Scatterplot with heat map . . . . .	8
2.4	<b>Evolution Radar</b>	An example of a software evolution visualization tool. . . . .	9
3.1	<b>CodeCrawler</b>	Package dependency view. . . . .	11
3.2	<b>XRay</b>	A screenshot of XRay System complexity view . . . . .	12
3.3	<b>XRay</b>	class dependency view . . . . .	12
3.4	<b>CodeCity</b>	A visualization of CodeCity made with CodeCity. . . . .	13
3.5		A screenshot of Chronia with a view of the ownershipmap. . . . .	14
3.6	<b>BugCrawler</b>	A screenshot of BugCrawler. . . . .	14
3.7	<b>Code Swarm</b>	A screenshot of code swarm's organic information visualization. . . . .	15
4.1	<b>General View</b>	A general overview of Lateralus . . . . .	17
4.2	<b>Visual Metaphor</b>	System view containing a list of the packages analyzed. . . . .	18
4.3	<b>Added Files metaphor</b>	Visual metaphor of added files. . . . .	18
4.4	<b>Lateralus 2D layout</b>	steps of the layout procedure of added files . . . . .	19
4.5	<b>Heatmap Example</b>	An USGS earthquake distribution on the U.S. . . . .	19
4.6	<b>Lateralus 2D</b>	2 dimensional gray-scale Heat map . . . . .	20
4.7	<b>Lateralus 3D</b>	Color scheme of the 3d Heat map. . . . .	20
4.8	<b>2D View</b>	2Dimensional view of a commit history in gray-scale mode. . . . .	21
4.9	<b>3D View</b>	View of a commit history in color mode. . . . .	22
4.10	<b>3D experimental view</b>	3 Dimensional experimental view of a commit history in colors. . . . .	22
4.11	<b>Lateralus</b>	behavioral process flow of Lateralus . . . . .	23
4.12	<b>Lateralus System view</b>	Class diagram of Lateralus. . . . .	24
5.1	<b>Nautilus</b>	The first 5000 frames of the nautilus commit history . . . . .	27
5.2	<b>Nautilus</b>	Animation frames from 5000 to 12'500 of the nautilus commit history . . . . .	28
5.3	<b>Nautilus</b>	Animation frames from 15'000 to 35'000 of the nautilus commit history . . . . .	29
5.4	<b>Nautilus</b>	Animation frames from 45'000 to 55'000 of the nautilus commit history . . . . .	31
5.5	<b>Nautilus</b>	Animation frames from 65'000 to 88'100 (final state) of the nautilus commit history . . . . .	32
5.6	<b>Epiphany</b>	The first 500 frames of the epiphany commit history . . . . .	33

5.7	The first 5000 frames of the epiphany commit history . . . . .	34
5.8	The frames from 7'500 to 23'500 of the epiphany commit history . . . . .	35
5.9	Animation frames at 30'100 of epiphany commit history . . . . .	36
5.10	The first 500 frames of the totem commit history . . . . .	37
5.11	Totem commit history frames 1000-6000 . . . . .	38
5.12	Totem commit history frames 8000-16500 . . . . .	39
5.13	Totem commit history frames 17100 . . . . .	40



# Chapter 1

## Introduction

Information is an elaborate process that attributes meaning to data sets, because of the complexity of the data and the attached semantics it is usually not easy to present in an effective way. Software systems can be regarded also as collections of large pieces of information. The development process of a software system creates a large number of data (the classes composing the system), thanks to visualization techniques there exists the possibility to explore this development process as an evolution of the system.

The challenge of making information easily available and understandable and to communicate to people the salient concepts, cannot avoid looking into the field of perception and cognition.

Therefore the main motivation behind this project is to understand how software systems evolve. As shortly discussed in the previously software systems have become complex artifacts that sometimes not even the creators are capable of understanding completely, thus to comprehend such a complex artifact it is necessary to reason in depth about the system. To think is the operation skill with which our intelligence acts (together with experience) for the purpose of understanding something.

A large number of articles have been published on the visual perception process, some of them concentrate the attention on the physiological foundations of our vision to try to infer more knowledge about the process of awareness and understanding in the context of visualization . The basis for the efficiency of information transmission associated with images, resides in the perceptive and cognitive processes of our brain[?].

### 1.1 Perception and Cognition

*"Visual organization is the deliberate prioritization of meaning within a visual design. Its the process of applying the principles behind perception - how we make sense of what we see - to illuminate relationships between content and actions." - Luke Wroblewski*

According to research evidence, visual senses have the predominant role in our capacity of sensing the world around us, what and how we see largely depends on how our brain acquires and processes the information gained through vision. Depictions of abstract concepts allow a visual sensorial experience, that assists the sense making process in complex situations.

The term perception is defined as the process in which a subject becomes aware of something through senses. Cognition on the other hand, is defined as the mental process of acquiring knowledge and understanding through senses.

In this work the idea is to apply principles of perception and cognition to a visual software analysis tool to gather an intuitive understanding of the development process of software systems.

## 1.2 Analytical Reasoning

When we try to understand something that we don't have knowledge about, a common mode of operation is to induce the sense making process by applying analytical reasoning techniques, these techniques help to structure information into understandable pieces that can be further processed and understood. Analytical reasoning techniques deal with an arbitrary set of conditions, such as the number of files in a system or the number of modifications and on the basis of the information presented, we try to make deductions about the relationships present in the system.

In this context visualization tools and techniques perform great as support analytical reasoning and are therefore very useful for gaining deeper understanding. Let's see how an image is useful to improve the understanding process. When we look at a depiction of something we get a first impression, when this happens our brain deals with this new information input keeping it in a buffer-like memory storage that is commonly called short term memory.

After an image has been processed by our short term memory, the brain can start working with new meaningful concepts that have been ordered and stored in our memory. Analytical reasoning techniques simply accelerate the understanding process of a complex subject giving intuitive knowledge of abstract concepts

The intention of visualization in this context is to improve the process of data presentation to support a deeper analytical reasoning about the subject being studied.

## 1.3 Three levels of Brain Vision

In the next section will present briefly, the way our brain deals with visual information inputs and how this information is processed. Researchers in the field of perception and cognition, like Colin Ware and Stephen Few, showed how our brain sees and processes the information around us. Common accepted theories in these fields state that we see with our brain at three different levels:

- **Iconic memory:** a specialized type of short term memory, it has a buffer-like recording capability that stores the images seen for less than a second before passing it to the short term memory. This kind of memory is able to store 3 to 7 elements for a very short time.
- **Short Term Memory:** also called primary memory, it allows to store small amount of information in mind for immediate availability of information for a short period of time (usually a couple of seconds). This memory is able to store from 5 to 9 elements
- **Long Term Memory** this is what we usually define as memory, namely the collection of our memories. This memory has unlimited capacity in terms of space and persistence.

## 1.4 Preattentive Processing

[?]Given our extraordinary cognitive abilities, it's incredible that all of this is achieved using short-term memory that can only hold from three to seven chunks of data at a time. This limitation must be considered when designing data presentations.

In this section I will present the basic concepts on which Lateralus is based, the idea behind the visualizations produced by the tool is to make leverage on the Iconic Memory and exploit the way it operates.

Preattentive processing is the operating skill which Iconic Memory uses, it is considered to be an iconic buffer in our brain, that keeps the images we see for approximately 250 millisecond before transmitting it to the short term memory, for further processing and sense making, this feature of our visual apparatus allows an instantaneous categorization of objects without conscious thought.[?][?]

The motivation behind Lateralus is that in order to improve the process of perception and cognition of the data being presented, a particular procedure taking preattentive processing techniques into account, needs to be used in order to present data to the user, in the most efficient way possible.



Figure 1.1: **Attentive Processing** An example of attentive processing

[?]Preattentive processing is an early stage of visual perception that occurs in iconic memory without conscious thought. It lets us rapidly recognize particular visual attributes that make certain things stand out or cause us to perceive visual groupings.

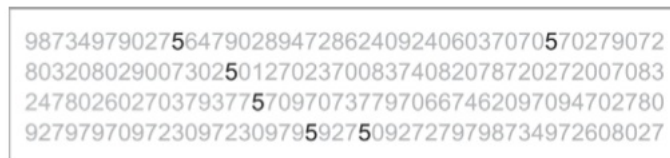


Figure 1.2: **Preattentive Processing** An example of preattentive processing

Preattentive processing refers to an organization of the depicted information based on cognitive procedures believed to be fast and performed at unconscious level, in other words these features allow certain symbols and shapes to be distinguished clearly among others due to their special attributes (called preattentive attributes) which will be discussed more in detail in the next section.

### 1.4.1 Preattentive Attributes

A researcher named Stephen Few, identified a list of primitive graphical elements that are perceived by our brain without conscious processing. Preattentive attributes are

primitive graphical elements and their relationships among each other that allow information to stand out without an intentional processing of these elements. The characteristic graphical features are listed below, they encompass the basic properties of every graphical depiction, namely hue and intensity.

[?]A preattentive attribute is a feature of complex information graphics that uses the cognitive feature of preattention to highlight the graphic's salient points in a manner that is most readily perceived by humans.

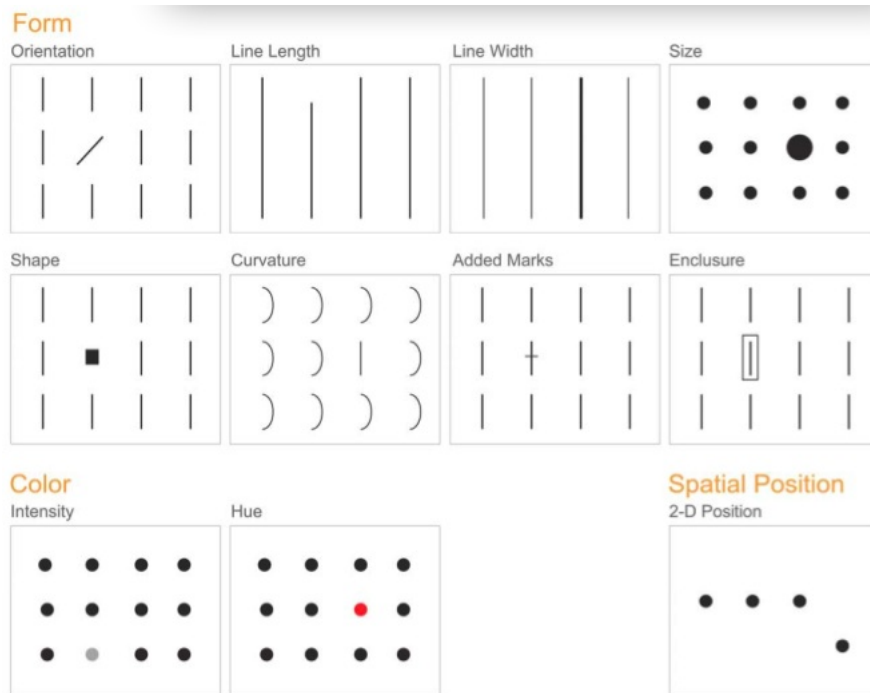


Figure 1.3: **Graphical primitives** Example of preattentive attributes.

The preattentive attributes are features that are particularly useful to categorize and discriminate shapes among a larger collection of objects that are similar to each other. These visual features of our optical system can be exploited to improve the process of data representation and improving the effectiveness of information visualization.

## Chapter 2

# Software Visualization

Information visualization is concerned with the study of the visual depiction of large sets of abstract information.

Complex software systems can be also considered as large sets of abstract data not easily understandable, the idea is to make use of graphical techniques to help the process of awareness about a complex construct such as computer programs can be.

Stephen Few makes clear why and how visualization can be helpful in the understanding process.

*"Why should we be interested in visualization? Because the human visual system is a pattern seeker of enormous power and subtlety. The eye and the visual cortex of the brain form a massively parallel processor that provides the highest-bandwidth channel into human cognitive centers. At higher levels of processing, perception and cognition are closely interrelated, which is the reason why the words "understanding" and "seeing" are synonymous. However, the visual system has its own rules. We can easily see patterns presented in certain ways, but if they are presented in other ways, they become invisible.... If we can understand how perception works, our knowledge can be translated into rules for displaying information. Following perception-based rules, we can present our data in such a way that the important and informative patterns stand out. If we disobey the rules, our data will be incomprehensible or misleading." - Stephen Few*

### 2.1 Historical overview

The idea of using images to depict concepts difficult to describe reaches far back into time, I will present some examples of how visualizing abstract concepts brought deeper understandings and provided an effective information vehicle.

#### 2.1.1 Petroglyphs

The first examples of depiction of complex actions can be considered the prehistorical cave drawings. Although the purpose of the paleolithic cave paintings is not known, some theories suggest that their was to transmit information, another theory instead suggests that these drawings might have a religious or ceremonial purpose, in both cases a picture could be much faster in transmitting information and be more descriptive than an oral explanation, even more so if the language is not evolved enough to vehiculate the information properly. Thus petroglyphs became the primary information



Figure 2.1: **Petroglyphs** Rock paintings from 30'000 years ago.

transmission vehicle.

35000 years ago when our ancestors wanted to communicate complex information about an elaborate action, like a hunt or tribal warfare, to the rest of his clan, he surely could have tried to describe it in words, but the prehistorical language almost certainly wouldn't have been evolved enough to communicate effectively every important aspect.

### 2.1.2 Maps and Historical Imagery

As mentioned before, the strength of information visualization resides in its fast transmission of complex information constructs, for example the depiction of an area of land or sea allows spatial understanding of a given area and helps orientation. Information of this kind are vital particularly in military, geographic and political contexts.

In military situations a map is a fundamental tool to allow strategists to get the big picture of the battlefield and position the troops and to take into account strengths and weaknesses of the terrain. Maps were and still are of extreme importance for navigation purposes, geographical representations are the origin and starting point of any geographical exploration

There exists a large number of different cartographic representations, road maps are the most widely diffuse maps nowadays, navigational maps with aeronautical and nautical charts, railroad network maps (see Figure 2.2). In addition to information about localization, maps can be used to depict additional information like elevation, temperature, rainfall, human density and so on.

### 2.1.3 Modern Scientific Visualization

Scientific visualization focuses on the use of computer graphics to create visual images which aid in understanding of complex, often massive numerical representation of scientific concepts or results.[3]

Scientific visualization applications are found in numerous fields from natural science, geography, ecology, formal sciences and applied sciences. The large diffusion that these techniques have in the scientific community, demonstrates the practical utility of visualization tools and techniques in conveying information rapidly. In figure 2.3 an example of scientific visualization.



Figure 2.2: **Historical Atlas of Europe** Historical map from the 16th century

In the early 90's visualization helped to achieve great breakthroughs and new understandings in scientific areas such as chemistry, biology and physics through molecule modeling, simulations of physical forces, DNA sequencing and weather phenomena to cite a few.

## 2.2 Applied Software Visualization

The purpose of visualization techniques, is to depict complex system with simple visual metaphors in order to facilitate the cognition of the data being presented. The most important feature of a visualization system should be to break down the complexity into an easier to understand system with an instantaneous information feedback. This tasks of visualization was clearly identified and described in a report for the U.S National Science Foundation, written in 1987 by B. McCormick.

Research evidence in the field of visual perception lead to the conclusion that confirms the hypothesis of the power of images as a medium to divulge information.

Information visualization is concerned with representing data with a visually speaking metaphor in order to obtain deeper understanding about symptoms that can help to spot and identify anomalies.

Due to technological development, visualization techniques can be applied to diverse fields ranging from aeronautical engineering through scientific and medical research up to applied science. Software engineering also benefits from visualization techniques, a branch of software engineering explicitly deals with the depiction of complex software constructs and analyzes their key features and their evolution though time.

In software systems, entities that can be analyzed to understand are files, on top of that other secondary entities can be used to gather information about the system being analyzed. Visualization techniques can give information about complex systems without a conscious thought about the system itself, it is therefore applied when an abstract or concrete concept that have to be described and understood are too difficult



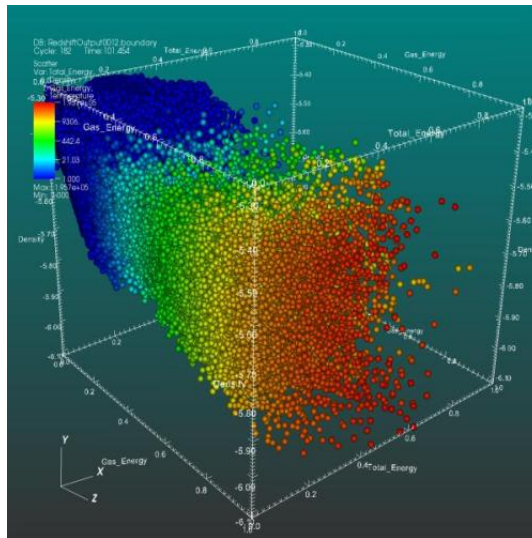


Figure 2.3: **Example of Scientific Visualization** 3D Scatterplot with heat map

or too expensive to convey in terms of time or words.

The results of applying visualization techniques to software showed to be a useful approach to comprehend the relationships and interactions of large software systems.

To understand better we can make an example, if we reason about the way we perceive the world, namely through the interaction of our vision and our brain, and we then consider how images speak to us more than complex textual or spoken descriptions we finally understand the real effectiveness of visualization tools especially when applied to software systems.

Software visualization deals with the visual representation of large non-numerical data sets, taking into account entities like files and lines of code in software systems. It is intended to give information about the system being depicted through visual feedback without having the developer look at the source code.

The goal of applying research evidence from the fields of perception and cognition is to improve the analytical reasoning process and as consequence a gain deeper understanding of the system in its completeness.

As mentioned before, there is a scientific motivated inspiration for the choice of analyzing a system through the visual senses, since software visualization amplifies the cognitive potential through the increase of the cognitive resources at disposal for analysis.

Software visualization increases the cognitive resources, this is accomplished by reducing search efforts, enhancing the recognition of patterns and sustain the monitoring of large numbers of events to spot relationships otherwise difficult to grasp.

”Scientists need an alternative to numbers. The use of images is a technical reality nowadays and tomorrow it will be an essential requisite for knowledge. The ability of scientists to visualize calculations and complex simulations is absolutely essential to ensure the integrity of analyses, to promote scrutiny in depth and to communicate the result of such scrutiny to others... The purpose of scientific calculation is looking, not enumerating. It is estimated that 50% of the brain’s neurons are associated with vision. Visualization in a scientific calculation is aimed at putting this neurological machinery



to work”.[?]

It has been observed that making an optical depiction of a complex construct, improves short term memory dramatically. Another positive feature that arises is the ease in the recognition of patterns when information is organized in space, taking into account its time relationships.

According to J.J Thomas and K.A. Cook, there are six basic ways in which we can experience an increase in the cognitive resources at our disposal, in their book, they are identified as follows:

by increasing cognitive resources, such as by using a visual resource to expand human working memory, by reducing search, such as by representing a large amount of data in a small space,

by enhancing the recognition of patterns, such as when information is organized in space by its time relationships,

by supporting the easy perceptual inference of relationships that are otherwise more difficult to induce

by perceptual monitoring of a large number of potential events, and

by providing a manipulable medium that, unlike static diagrams, enables the exploration of a space of parameter values.[?]

The positive features of visualizations lie in the power of representing a large amount of data in a small space making information patterns and relationships pop out. Thus through the depiction it is possible to modify the understanding of a complex and abstract construct

## 2.3 Software Evolution Visualization

Visual software evolution analysis takes advantage of research evidence in the field of perception, cognition and information processing in order to improve the insight and sense making about the system being observed.

[?]The goal of software evolution research is to use the history of a software system to analyse its present state and to predict its future development.

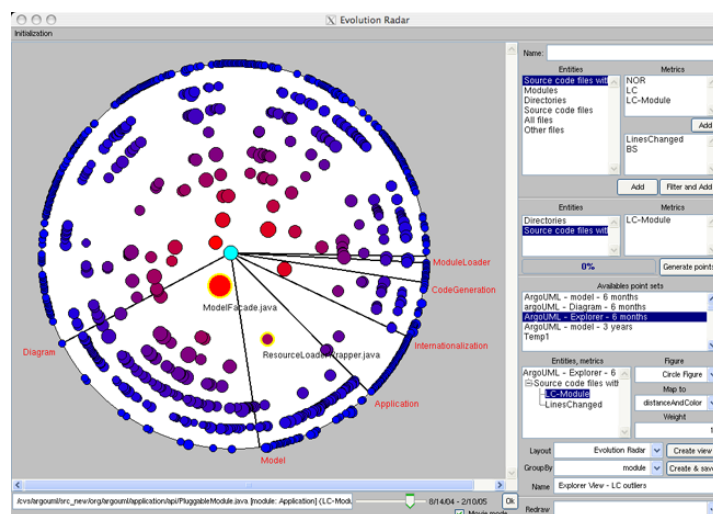


Figure 2.4: **Evolution Radar** An example of a software evolution visualization tool.

Many software systems are built by larger communities. Collected data shows that during the development process, each developer may change substantially the system with its modifications. Each of these changes define new revisions of the project, and if considered useful they become part of the system. In this sense a software development process has traits that resemble those in human evolution. In the theory of evolution genetic alterations that result in better adaptation to the environment are preserved, the same approach exists in software development and constructs are modified through time keeping the modifications that suit at best. As we have seen the central aspect of evolution of software systems is related to changes, in ?? a screenshot of Evolution Radar 2.4 written by Marco D'Ambros, a tool that deals with the evolution of software systems. With Lateralus the focus is on the analysis of these modifications and try to produce a visual interpretation in order to infer additional information about the subject at hand. In the next section I will present some examples of how visualization tools can broaden the understanding of software systems.

# Chapter 3

## Related Work

### 3.1 Software Visualization Tools

In this section I presented some related research work that has been done in the area of software visualization in general and the evolution of software in particular. There are several tools that deal with code visualization, here are two examples.

#### 3.1.1 CodeCrawler

CodeCrawler is a language independent reverse engineering tool which combines metrics and software visualization. CodeCrawler is based on Moose, a reengineering environment developed by members of the Software Composition Group. CodeCrawler is written by Michele Lanza in VisualWorks Smalltalk and runs on every major platform.[?] As you can see in 3.1 CodeCrawler uses a graph based view to depict entities and their relationships.

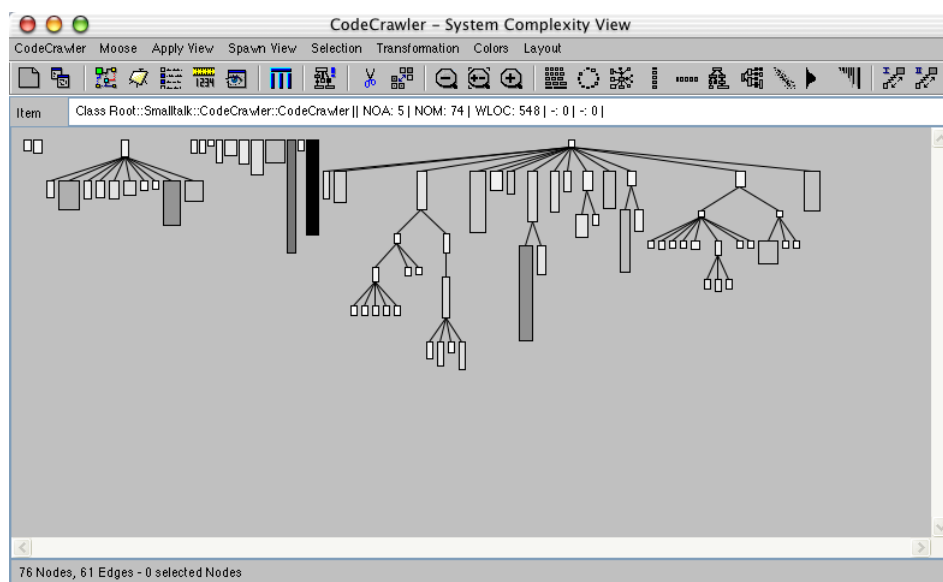


Figure 3.1: **CodeCrawler** Package dependency view.

### 3.1.2 X-Ray

X-Ray is an open-source software visualization plug-in for the Eclipse framework. This tool was written by Jacopo Malnati as bachelor project . It provides System Complexity View, Class and Package Dependency View for a given Java project. Moreover, its model of the underlying Java project can be triggered and used by other plug-ins.[?] In 3.2 and fig-xray2 views of X-Ray

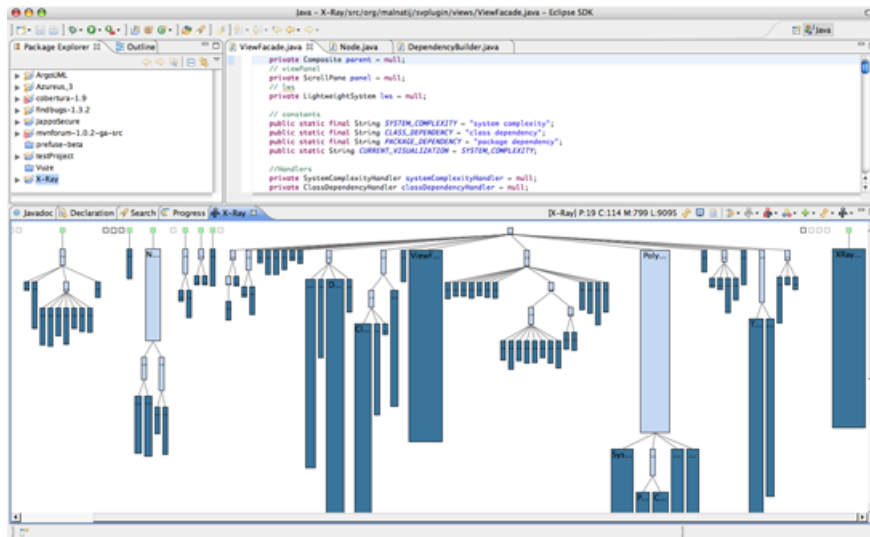


Figure 3.2: **XR**ay A screenshot of XRy System complexity view

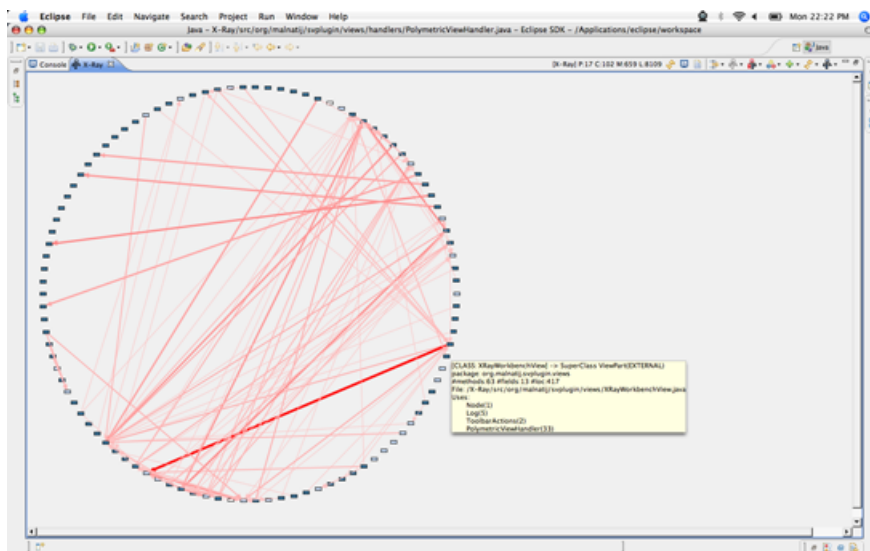


Figure 3.3: **XR**ay class dependency view

## 3.2 Software Evolution Visualization Tools

The following examples refer to visualization tools which focus on the evolution of software through time.

### 3.2.1 CodeCity

CodeCity was written in 2006 by Richard Wettel, a PhD student at the Informatics Faculty of Lugano. CodeCity, is a language-independent interactive 3D visualization tool for the analysis of large object-oriented software systems. Using a city metaphor, it depicts classes as buildings and packages as districts of a software city. CodeCity was programmed in VisualWorks Smalltalk on top of the Moose platform, uses OpenGL for rendering, and runs on every major platform. [?]

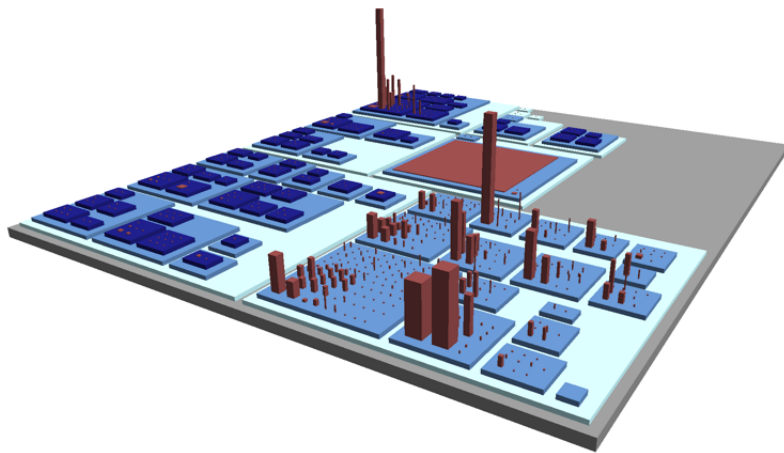


Figure 3.4: **CodeCity** A visualization of CodeCity made with CodeCity.

### 3.2.2 Chronia

Chronia is a tool to explore the CVS history of a software system, featuring a visualization that shows who owned which files at which time in a systems evolution. Chronia has been written by Mauricio Seeberger in 2006.

To understand a certain issue of the system it is possible to ask knowledgeable developers but, in large systems, not every developer is knowledgeable in all the details of the system. Thus, with this tool it is possible to know which developer is knowledgeable in the issue at hand. The Chronia ownershipmap allows to connect developers to files they own, making it easy to spot who owns which file[?], in 3.5 a screenshot of Chronia

On the visualization above, the X-axis is time and on the Y-axis are files. The circles are commits and the colors show authors. When a line is colored, at that time, most of the file was written by the according author (ie code ownership).

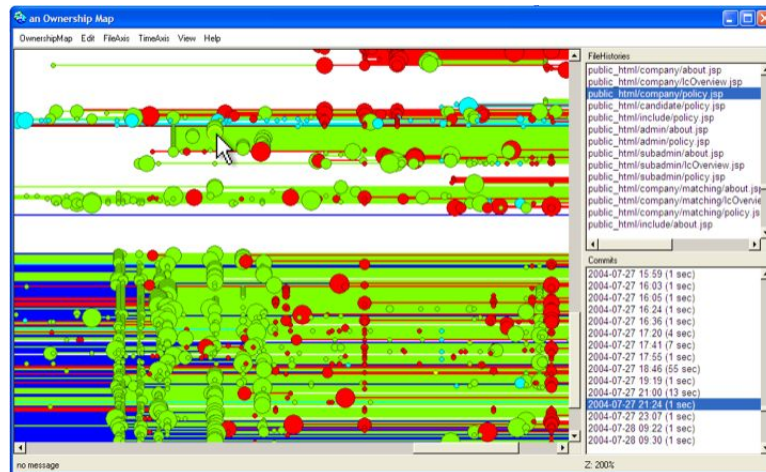


Figure 3.5: A screenshot of Chronia with a view of the ownershipmap.

### 3.2.3 BugCrawler

BugCrawler is a tool developed to track the lifetime and see the evolution of software bugs, these experience usually a long existence, persisting through different stages of the development process. [?][?] As explained above, this tool is meant for the analysis of software bugs, it was written by Marco D'Ambros in 2006 and it is capable of tracking large software systems and analyze the evolution of bugs affecting the system as you can see from the screenshot in 3.6. A short description of the tool given from the

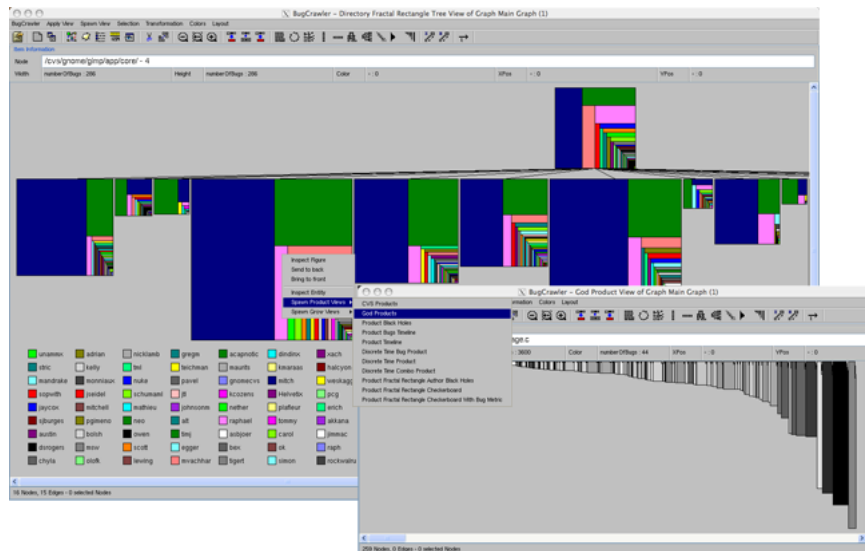


Figure 3.6: **BugCrawler** A screenshot of BugCrawler.

author:

*"BugCrawler is a tool written in Smalltalk for analyzing the evolution of large software systems. The tool is based on interactive visualizations and software metrics. The information used for rendering the views is based on the software system itself, CVS logfiles and Bugzilla problem reports."*



# Chapter 4

## Lateralus

### 4.1 Idea

Software visualization systems typically represent software metric data gained from reverse engineering or measurements of the systems.

Lateralus goes under the category of software evolution visualization, it has been written as a combination of Java and Processing, and allows to gather additional information about the system through the analysis of its commit history.

The motivation behind this project is to understand better a software system during its development, the goal to achieve this was to build a tool capable of analyzing the growth of a software construct based on its Subversion commit history.

The Lateralus tool analyzes an SVN commit history and uses it to gather information about a system and discover anomalies and patterns arising during the development of a software across time. This tool is intended to give a visual interpretation to changes, related to the development of a project, and monitor the evolution of a software system through time, based on the log file of the SVN versioning system.

The idea behind this tool is to couple data and its representation with elements that are easily identifiable without a conscious thought, by using theories developed in the field of perception and cognition research. I'll present some theories that are the ground assumptions on which this tool has been built. By adopting principles of visual perception and cognition coupled with the use of preattentive processing attributes in visual software analysis, this tool produces visual feedback that allows to monitor anomalies in intuitive way, through characteristics of the graphic elements such as position and color.

The goal is to depict the evolution of the development stages of a project, by exploiting analytic reasoning and principles of visual perception and cognition and through this approach to produce a visual feedback that helps and supports information processing.

### 4.2 Overview

The use of software versioning system is a standard requirement in large or geographically scattered development communities in order to maintain a productive development environment. In order to monitor the evolution of the system, its svn commit history is analyzed and processed. This tool takes a distinctive feature of a file (like the number of modifications), represents it and change its position or color attribute based on its age and the degree of modifications, this allows to easily spot anomalies and outliers without looking at the source code. Below in 4.1 a schematic view of the general structure



of Lateralus, as you notice, we gather an svn commit history from a repository, parses and analyzes its trace, Lateralus then creates the object and renders the frames to the screen.

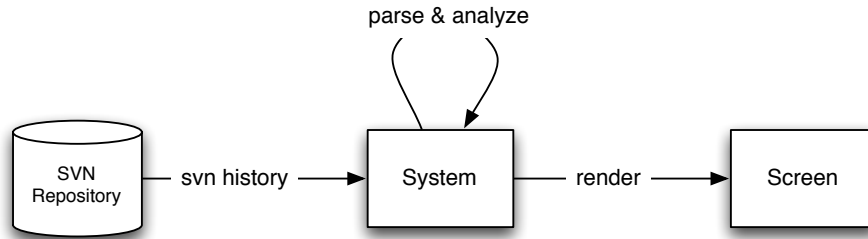


Figure 4.1: **General View** A general overview of Lateralus

## 4.3 Features

Lateralus is a tool that creates a visual animation of the commit history of an SVN repository, moreover Lateralus allows to visualize the history of the system based on the changes made to it. The goal of supporting the analytical reasoning process is met with the production of images depicting the different stages of development, in addition Lateralus is able to take snapshots with any sample period, Lateralus is also capable of storing the images for further analysis. Other features include basic user interaction with the animation to infer additional information about the system, and more specifically about the files being changed (time stamp of changes, author of changes, etc)

## 4.4 Visual Metaphors

As described in the previous section, the idea is to associate a visually speaking metaphor to the entities being analyzed (in this case files and their modifications), taking into account principles of visual perception and theories of cognition.

The pieces of information that are available from an SVN repository are limited. The basic entities are files being added to the project. These files can be modified and deleted by different developers at different moments in time. The authors and time stamps of the modifications of the system are stored as entries in the SVN trace, forming the commit history of the project.

All this data does not bear additional information in its unstructured form, it needs to be mapped into visually speaking metaphors in order to take advantage of the power of images as a vehicle of information.

### 4.4.1 Abstraction and Rendering

The abstraction presented in Lateralus maps the files composing a software system to a simple graphical element, that can be either a two dimensional square or a three dimensional cube (as in 4.2), each of these shapes represents a file that has been added.

Moreover changes to the system have been represented with the metaphor of a heat



Figure 4.2: **Visual Metaphor** System view containing a list of the packages analyzed.

map, where the color hue increases its warm tones whenever the file is modified, and decreases its hue to colder tones when the file is left unaltered.

#### 4.4.2 Visualizing Files: Squares and Cubes

The chosen metaphor to depict the files entities are simple graphical elements such as squares for the 2Dimensional representation and cubes for the 3Dimensional representation, these shapes have been chosen due to their simple form, this simplicity should suppress any distraction from the figure itself.



Figure 4.3: **Added Files metaphor** Visual metaphor of added files.

The depiction of the chosen abstraction presented above in 4.3, shows an added file that has not yet been modified. The position and the color of these primitives shapes give additional informations about the file subject, its position reveals the age while its color tells the degree of modifications with respect to other files. Modifications of files over time will also change the color of the square representing the file being changed, in the initial state an added file will have a pale-blue color to symbolize that it is cold (meaning that it hasn't been modified yet)

#### 4.4.3 Visualizing Age: Layout and Position

The metaphor chosen for the layout was inspired by a previous work of prof. Lanza where the spiral positioning of shapes was used to make the age of the files easily identifiable through their spatial position. The placement of the shapes follows a spiral form, with the oldest files at the center, and the more recently added files on the outside of

the spiral. Through the use of this metaphor, it is always clear which are the oldest core components of the system (represented in the center of the spiral). With such a simple metaphor it is easy to understand which are the recently added files and which are the older components, moreover this layout choice maximizes the field of vision, leaving the older files more centered than recent additions, but both in the field of vision without excluding parts of the system. In 4.4 an example of how the layout function places the shapes on the canvas.

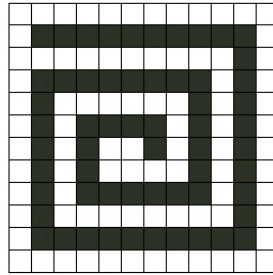


Figure 4.4: **Lateralus 2D layout** steps of the layout procedure of added files

#### 4.4.4 Visualizing Changes: Heat map

A heat map is a graphical representation of data where the values taken by a variable in a two-dimensional map are represented as colors.[?] In 4.5, an example of Heat map visualization applied to the U.S earthquake distribution

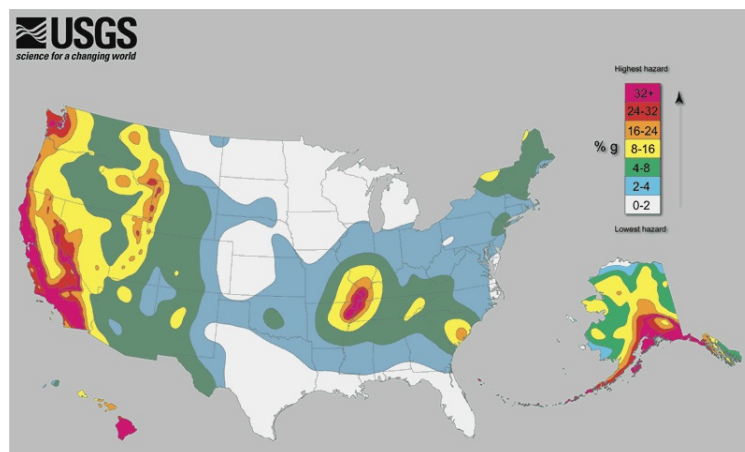


Figure 4.5: **Heatmap Example** An USGS earthquake distribution on the U.S.

The idea of depicting file modifications with changes to the color of the shapes, is inspired by the idea of a heat map, a visualization technique that puts a chromatic different layer on top of an image, this top layer has different colored areas to visualize the temperature differences, hot areas usually are associated with warm colors while

colder areas are associated to cool colors. This mapping can be used to connect different data sets, in Lateralus it is particularly useful to map the distribution of changes in files. The abstraction in this case was to map the number of changes to the temperature of the shape, making often modified files warmer and less often modified colder. Combining the heat map and spiral metaphors, offers therefore a solid visualization model, because of the intuitiveness and clarity in which the refactorings are depicted. For example deletion of files, an operation which usually is performed on relatively new files with little dependencies, result in holes in the spiral layout, thus showing the level of structure of the system and the dispersiveness of the files composing it.



Figure 4.6: **Lateralus 2D** 2 dimensional gray-scale Heat map

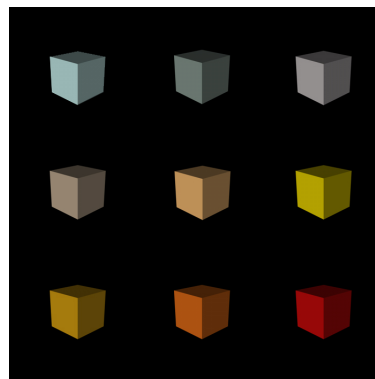


Figure 4.7: **Lateralus 3D** Color scheme of the 3d Heat map.

In 4.6 and 4.7 are presented the 2 dimensional and 3 dimensional views of the heat map and their associated color schemes. In the 2 dimensional view the visualization has 2 different variations of heat map, a grey-scale and a color heat map. Even though the first one (gray scale) was subjectively more visually attractive from my point of view, only the latter (color based) has been adopted in the validation process. This is due to the limitation of the greyscale heat map where the codification of changes to color cannot make use of the Hue property, it just modifies the intensity of the black color tone to make the differences visible.

This visualization is a three dimensional based view with the a simple heat map color scheme, where, the files being modified more often get a temperature increase of their surface, which is mapped to a warmer tone in the color intensity, so a red cube has

been modified more often than a pale-blue cube, alike in the gray-scale color scheme, darker tones correspond to "heat" and therefore frequently modified files, while lighter tones correspond to less modified entities.

## 4.5 Views

Lateralus has different visualization modes a two dimensional and a three dimensional view. The two dimensional view allows to focus better on the changes in the system and spot anomalies easier, on the other hand the three dimensional view allows to differentiate the spatial understanding and the age of files within the system Below I will present some screenshots of the different views available in Lateralus.

### 4.5.1 2D View

The two dimensional view below in 4.8 allows a rapid recognition of shapes and colors, the loss of the 3rd dimension helps to focus on the color intensity changes rather than on the positioning of the shapes. The shapes can be modified and manipulated by two-dimensional geometric transformations such as translation, rotation and scaling.

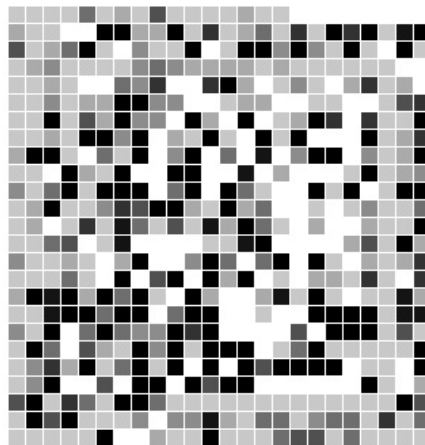


Figure 4.8: **2D View** 2Dimensional view of a commit history in gray-scale mode.

### 4.5.2 3D View

The 3D view (see 4.9) allows to to spot files being modified in a three dimensional environment, this allows a better spatial understanding of the system being depicted. 3D models represent a 3D object using a collection of points in 3D space. Since these models are a collection of data (position, age, number of changes and other information) they need to be represented on the canvas through an OpenGL renderer.

### 4.5.3 Experimental 3D View

This is a view that unfortunately is not yet part of Lateralus, it is based on a library that allows to produce CAD models in Processing, below you can see a screenshot of

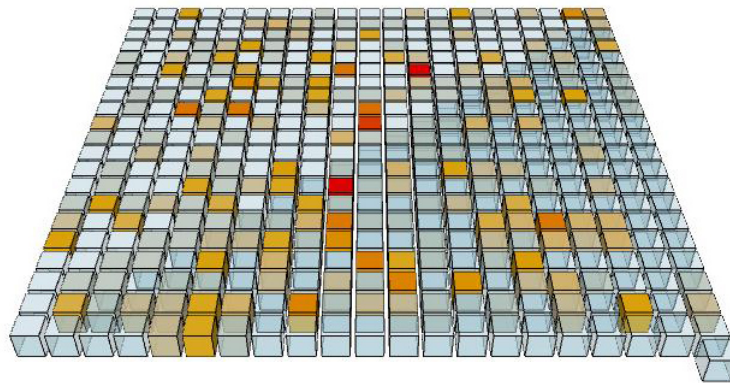


Figure 4.9: **3D View** View of a commit history in color mode.

this experimental view, it makes use of an external library called Anar+. This library is extremely powerful, but due to the lack of time at disposal to work with it, I was unable to fully integrate this view in the project. Although there are some problems to integrate this library with the existing code, the visual feedback obtained from the use of this library kit is the very appealing and would allow much better interaction with the 3D space than the actual 3D view. Below in 4.10

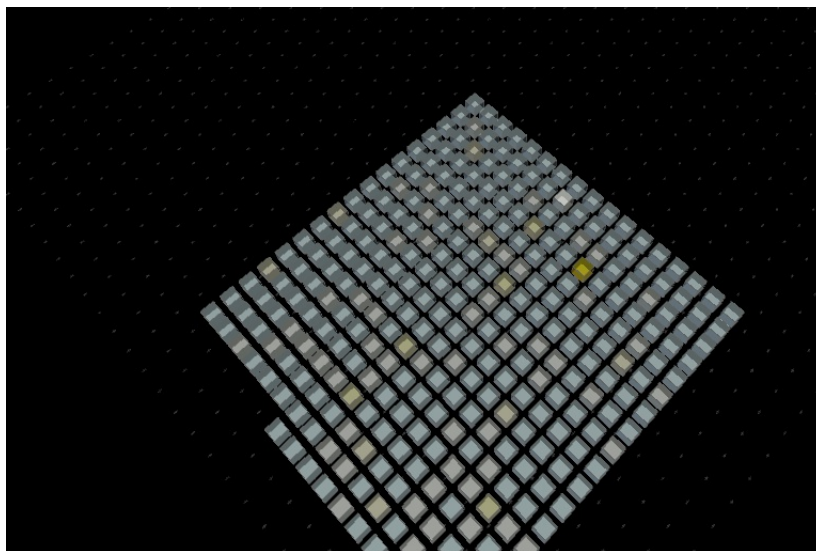


Figure 4.10: **3D experimental view** 3 Dimensional experimental view of a commit history in colors.

As mentioned before, due to the late discovery of this collection of libraries, it was impossible to make an extended use of it. The possibilities to produce a 3D visualization with a better interaction with the environment would benefit a lot from the features of this library kit. The large number of available features makes this library kit an

extremely powerful and promising extension that probably would have been the best choice to start with.

## 4.6 Implementation

The following diagrams depict the sequence of operations that the system undergoes during runtime. The figure additionally shows the setup preprocessing and realtime processes performed.

### 4.6.1 Behavioral View

This is a general scheme of the behavior of Lateralus .

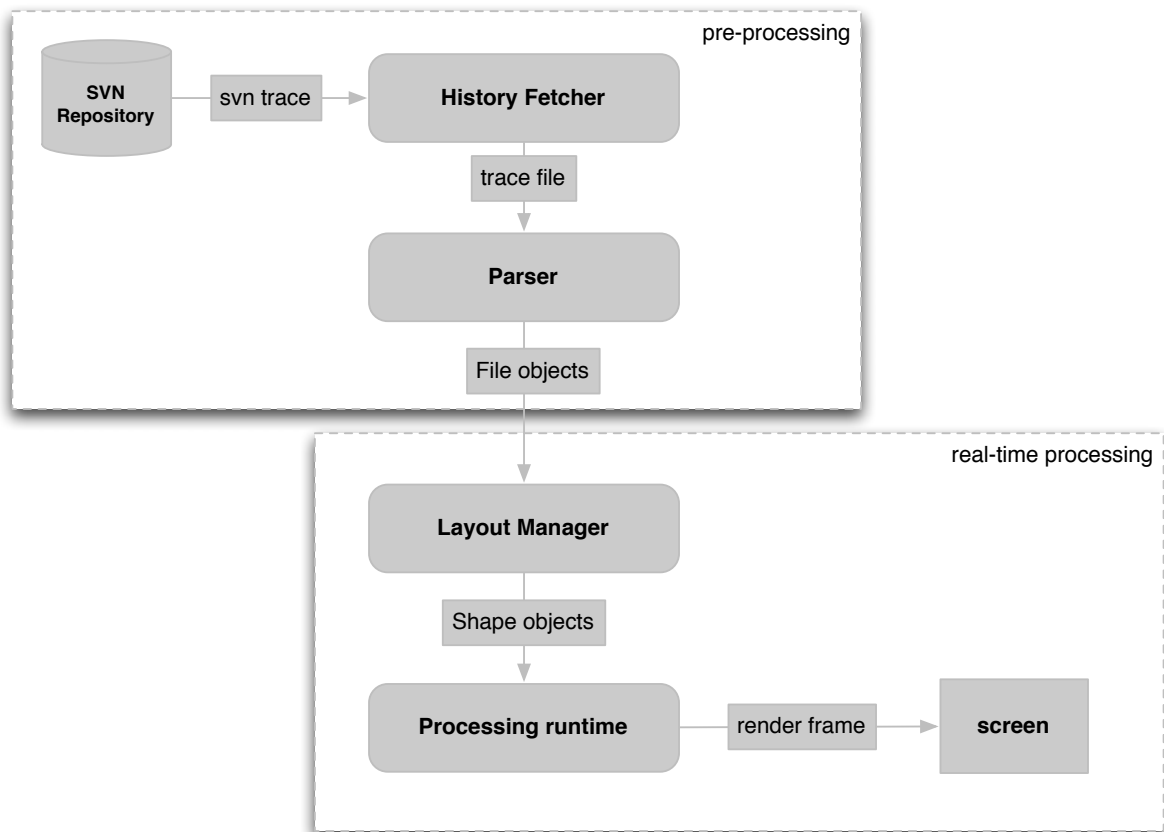


Figure 4.11: **Lateralus** behavioral process flow of Lateralus

The tool connects locally or via the internet to an SVN repository through the HistoryFetcher module and analyzes the repository content, it produces a trace of the commit history. The Lateralus parser reads the trace file and generates the visual metaphors of the files, the LayoutManager then takes care of the placement of the shapes on the canvas, finally Lateralus renders the frames that depict the system and at runtime produces an animation of the development behavior from a file-change based perspective.

## 4.6.2 Structural View

The class diagram in 4.12 depicts the general structure of the system. To notice how the system architecture follows the MVC pattern to maintain separation of concerns and independent testability.

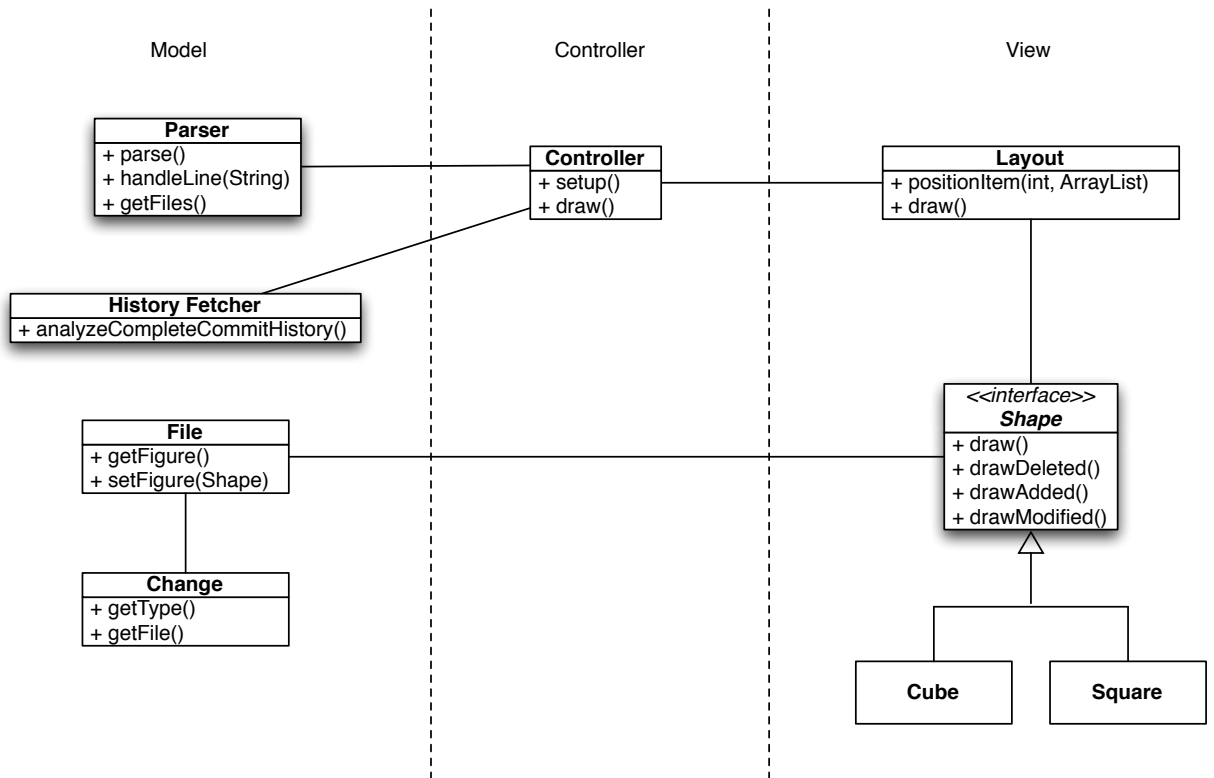


Figure 4.12: **Lateralus System view** Class diagram of Lateralus.

## 4.7 Components

Listed below, I'll present the necessary elements that have been used in the development and validation process of the Lateralus tool.

### 4.7.1 Subversion SVN

Subversion is a free version control system which operates similarly to CVS but with an extended feature set. Subversion is an industry standard in the software engineering community for developing large systems with many developers working.

### 4.7.2 Processing

Processing is an open source project developed by Casey Reas and Benjamin Fry, both formerly of the Aesthetics and Computation Group at the MIT Media Lab. It is a java



based programming language and integrated development environment , which focuses on programming in a visual context. The language builds on the graphical capabilities of the Java programming language, sby adding new graphical features to extend the scope of Processing. Processing is free to download and available for GNU/Linux, Mac OS X, and Windows.[?]

### 4.7.3 External libraries

In order to produce different visualizations it was required to introduce the following libraries:

#### **OpenGL**

[?]OpenGL (Open Graphics Library) is a cross-platform graphics interface for 3D and 2D graphics. This library allows Processing programs to utilize the speed of an OpenGL accelerated graphics card. This expands the potential for drawing more to the screen and creating larger windows. Processing interfaces with OpenGL through JOGL, an initiative from the Game Technology Group at Sun.

#### **Anar+**

Anar+ is a cadKIT for Processing (v1.0) for Object Oriented Geometry.[?] This collection of libraries are based on (anar+) parametric modeling scheme. The features of this library are listed below: Geometric Associativity (aka Parametric Modeling) Scenegraph for processing Geometric datastructure (groups, objects, faces, lines, points) Extended Geometric manipulations Objects based modular renders Export for various CAD formats (though metaScripts) Camera, view and walkthrough ...

# Chapter 5

## Lateralus Validation

In order to validate Lateralus, some test cases have been defined and fed into Lateralus. One of the necessary requirements to perform the validation process was to have several Subversion repositories and a possibly large developer community, the GNOME project, a Linux desktop environment that has a quite large developer community and an discrete amount of applications, seemed to be an ideal source for collecting Subversion repositories to be analyzed.

### 5.1 GNOME Applications

The GNOME applications taken into account were picked randomly in order to have a broad sample that would show a difference in the results of the analysis. Some of the chosen applications were picked in reason of their size or for their peculiarities in terms of functionality, below a reduced list of some of the repositories observed closer.

- Alacarte (Menu editor)
- Epiphany (Web browser)
- Gedit (Text editor)
- Nautilus (File manager)
- Evolution (File server)
- Seahorse (Encryption Key manager)
- Tomboy (Note taking software)
- Totem (Media player)
- GNOME desktop (the desktop environment)

The complete list of repositories is in fact larger but the number of possible candidates for validation had to be restricted to the most meaningful and interesting results.

#### 5.1.1 Test Case 1: Nautilus

Nautilus is the official file manager for the GNOME desktop. Its repository had more than 89'000 entries. Due to the static nature of this paper it is very difficult to convey the animated visual feedback properly, below I show a the evolution of the nautilus commit history, based on a set of screenshots taken with Lateralus, unfortunately due to the large number of pictures taken, I'll present the status of nautilus every 500

frames and enlarge the period of sampling as the number of files increases.

We can observe in the following screenshot in fig 5.1 of the Nautilus evolution, how the developers of the system soon started to files and modified them to add functionalities, in particular the developers work is visible at the center of the spiral (in particular the shots at 200 frames and at 500 frames), in the next phase (from frame 1000 to 2000) the core functionalities start to be defined. The enclosed yellowish square at frame 2500 already shows what part of the system is going to be the core functionality. It has been observed that the development cycle undergoes a 6-month release period for the developer community. This translates in explosion-like addition of many files in short amount of time. It is easily observable that many added files, get then discarded as the evolution of the system proceeds.

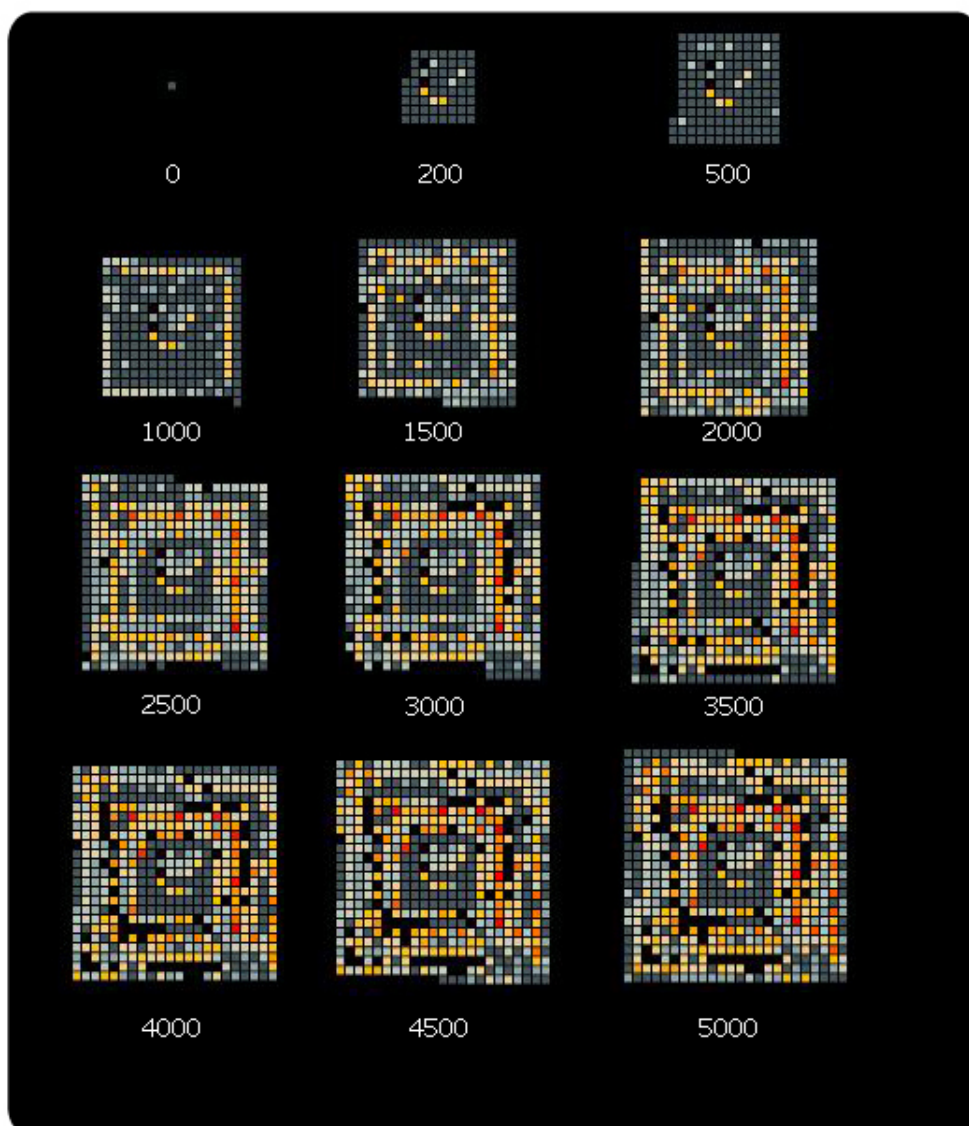


Figure 5.1: **Nautilus** The first 5000 frames of the nautilus commit history

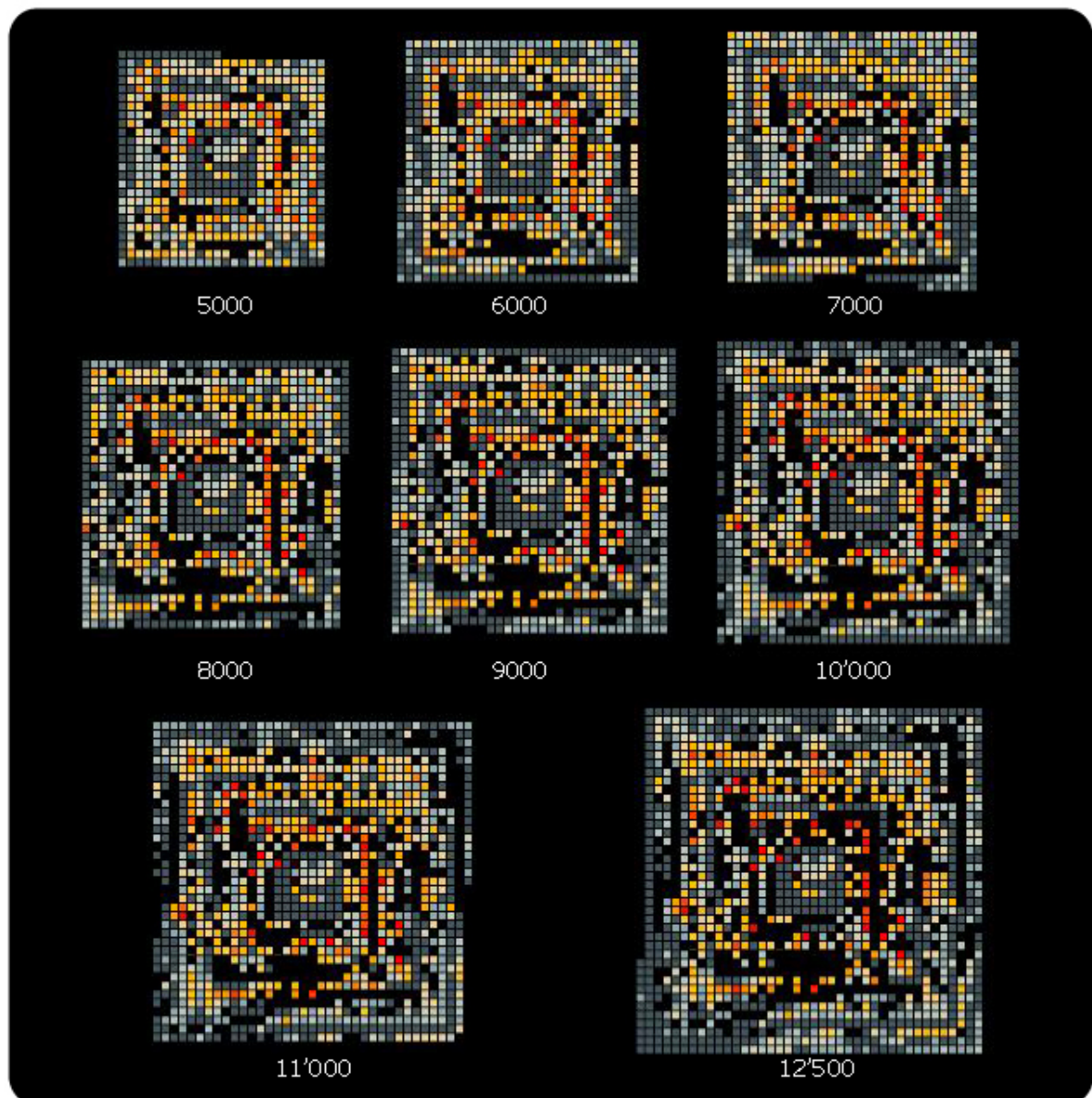


Figure 5.2: **Nautilus** Animation frames from 5000 to 12'500 of the nautilus commit history



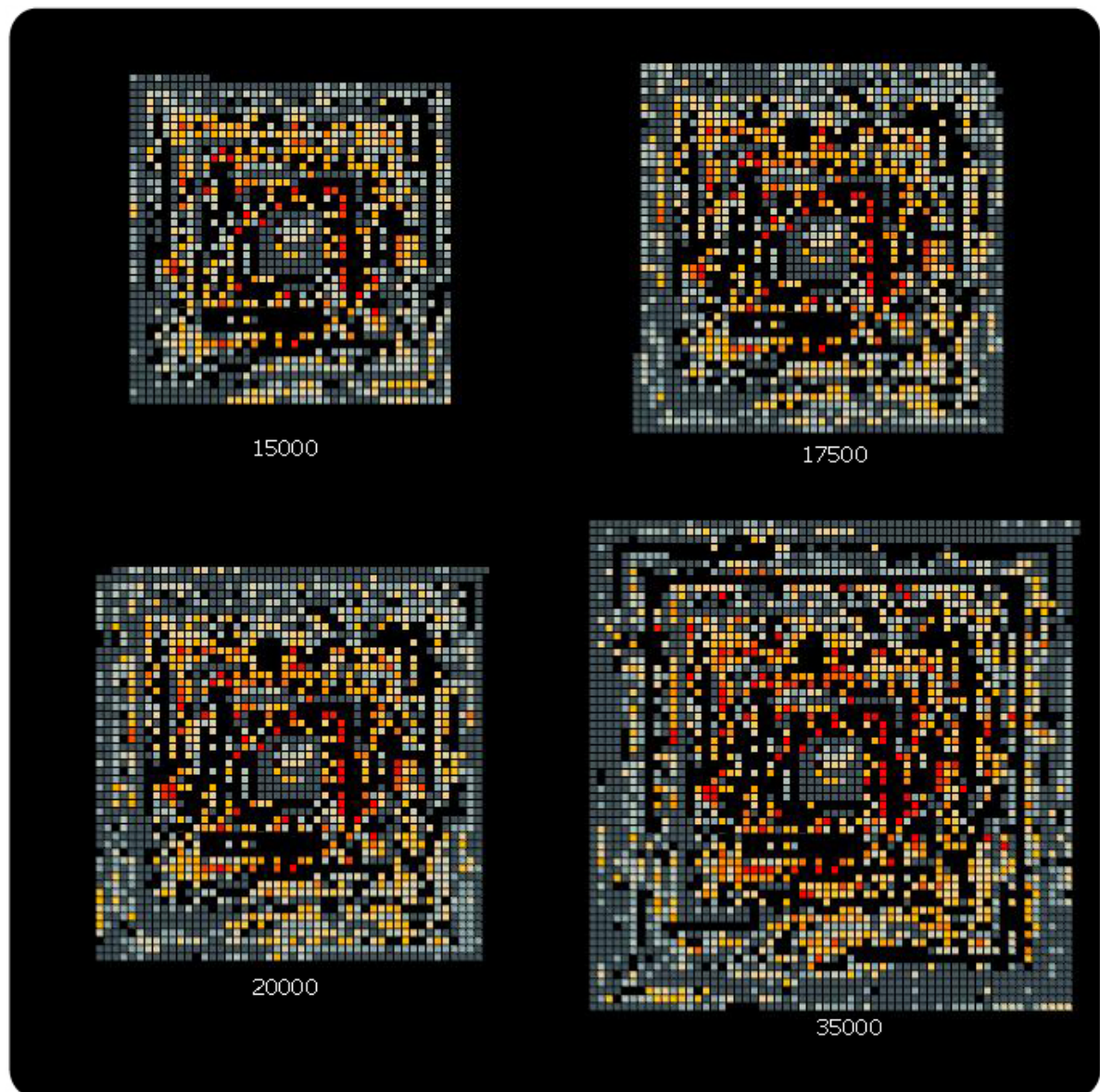


Figure 5.3: **Nautilus** Animation frames from 15'000 to 35'000 of the nautilus commit history

We can clearly see that there is an addition of files that are not being modified in the future, these files might be created for the purpose of helping the core functionalities, that works as a sort of infrastructure for the system itself. We also may observe that the Pareto rule, that states that 80% of the functionality depends on only 20% of the files in the system, is proven to be correct, as we can see after the deletion of many files the amount of files that get a red tone is limited when compared to the overall number of files. In fig 5.5 we can see the final state of Nautilus. Many files have been deleted, but the functionality identified previously is still present, these files have been modified extensively, making them the core of the system.

### 5.1.2 Test Case 2: Epiphany

Epiphany is the official web browser for the GNOME environment. Its repository had around than 30'000 entries. It is the official browser in the GNOME community and acts as an excellent test case for this validation. This system differs from the previous one, since it has a late start in introducing the core functionality. In fig 5.6 we can see that many files have been added and left untouched, almost 300 files have been added before starting to work on the core functionality of the system. Still in fig 5.6 at frame 500, it is possible to see the first changes to the system, many files change color to yellow tones, signaling a discrete amount of modifications.

In fig5.7 at frame 1000 we can observe how some of the added files have been removed. In the following frames it is visible how the core functionality is more spread among the system with respect to the previous test case (nautilus). As we see in fig 5.8 the system has more modified Files, this is clearly made visible by the increasing red tone of the image. In fig. 5.9 we can see the final state of the epiphany system, in this stage it becomes more similar to the previous analyzed system. With a set of core functionalities (in red), that happen to be the some of the oldest files of the system, and a wider set of infrastructure files (yellow) that support the whole system.

Again we can identify the Pareto principle, if we estimate the amount of core functionality we realize that it happens to be around the 20% of the total number of files composing the system.

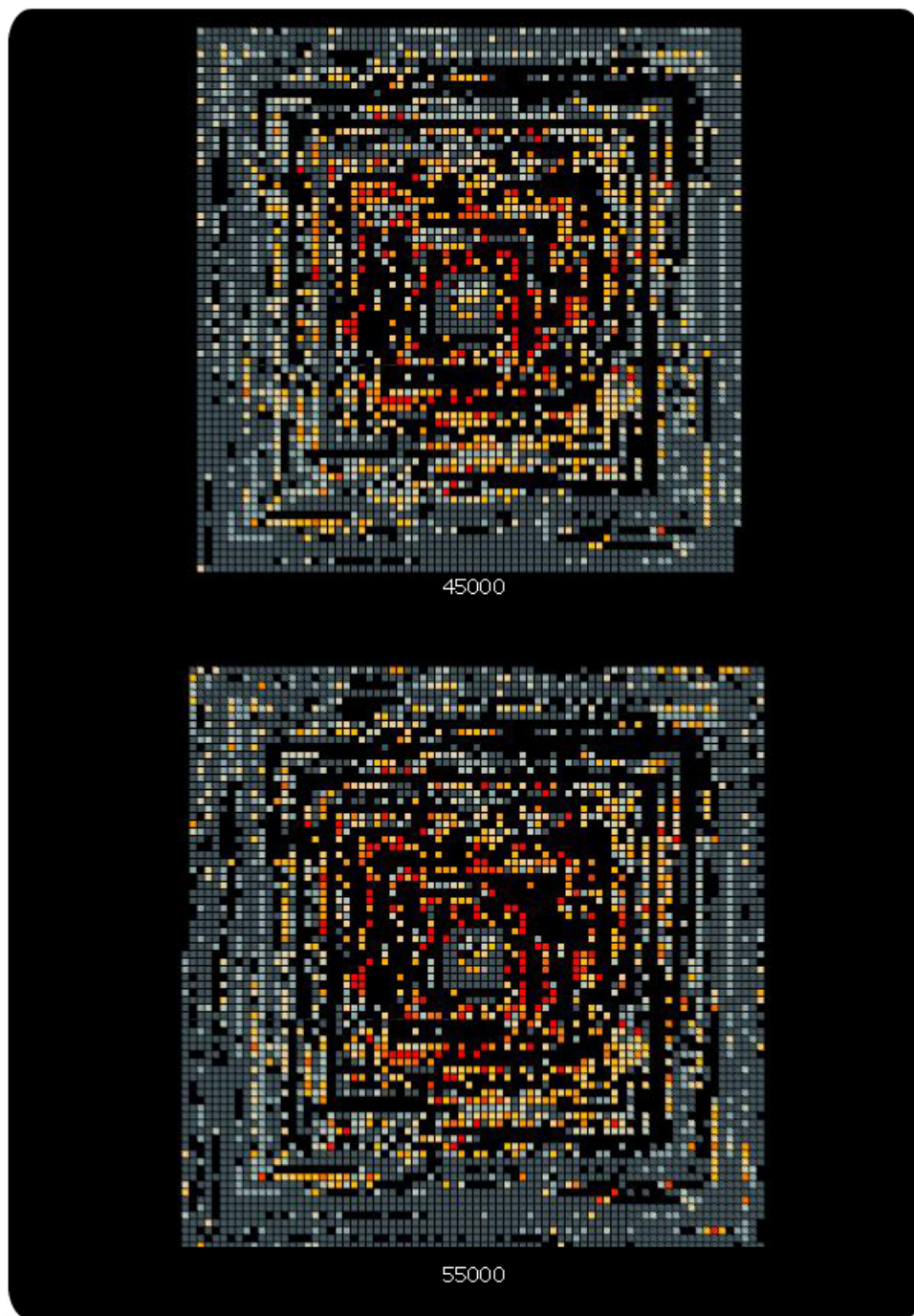


Figure 5.4: **Nautilus** Animation frames from 45'000 to 55'000 of the nautilus commit history



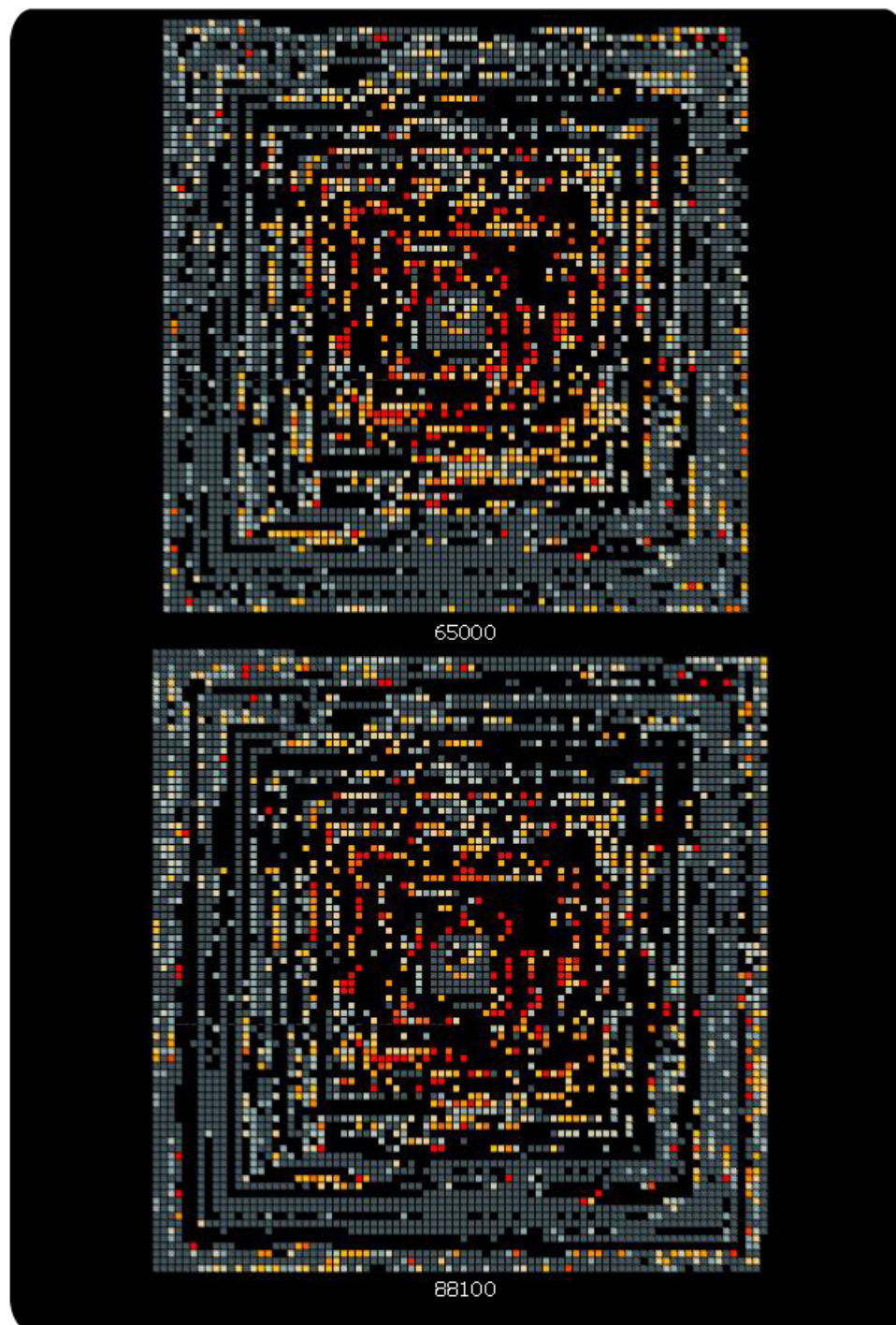


Figure 5.5: **Nautilus** Animation frames from 65'000 to 88'100 (final state) of the nautilus commit history



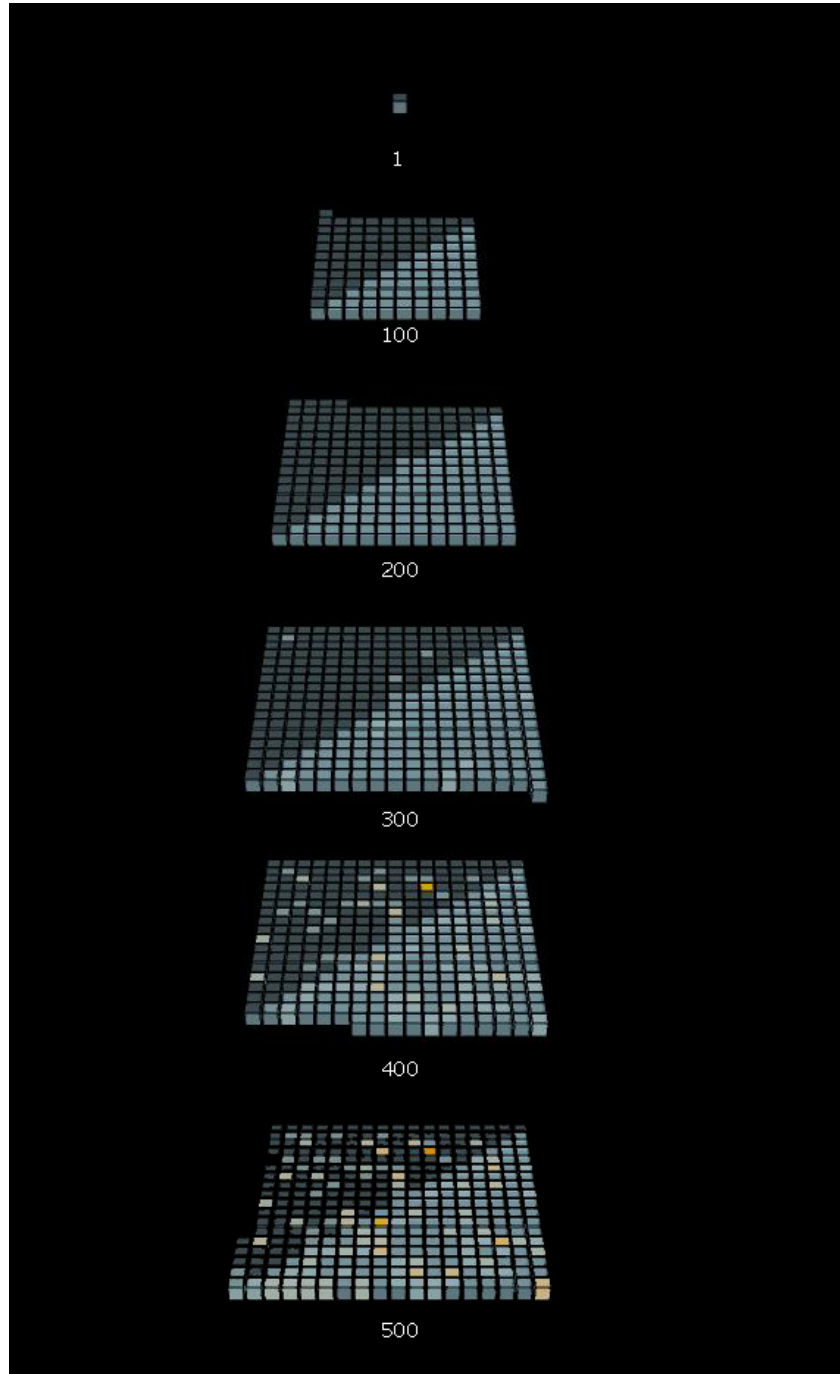


Figure 5.6: **Epiphany** The first 500 frames of the epiphany commit history

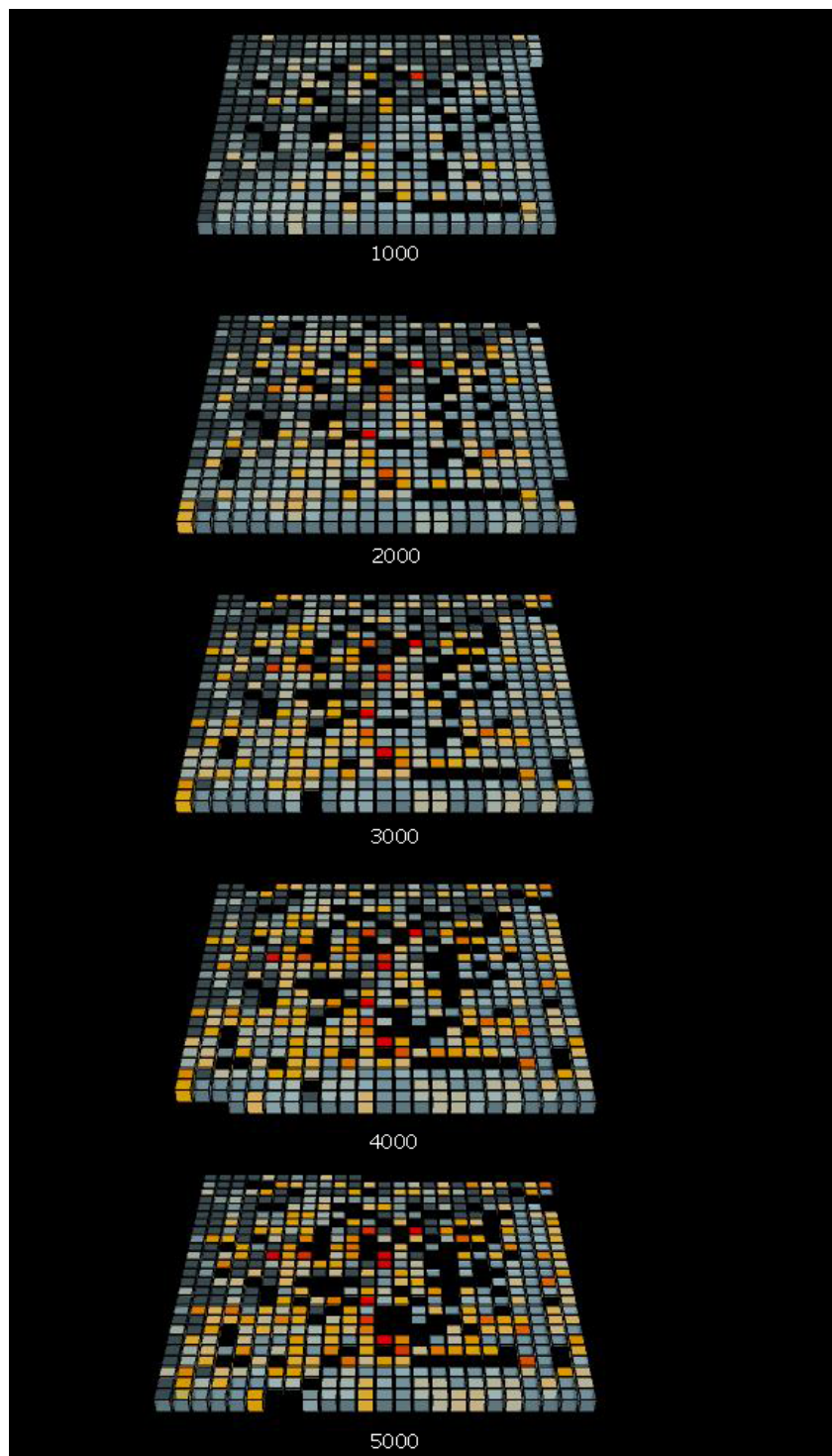


Figure 5.7: The first 5000 frames of the epiphany commit history

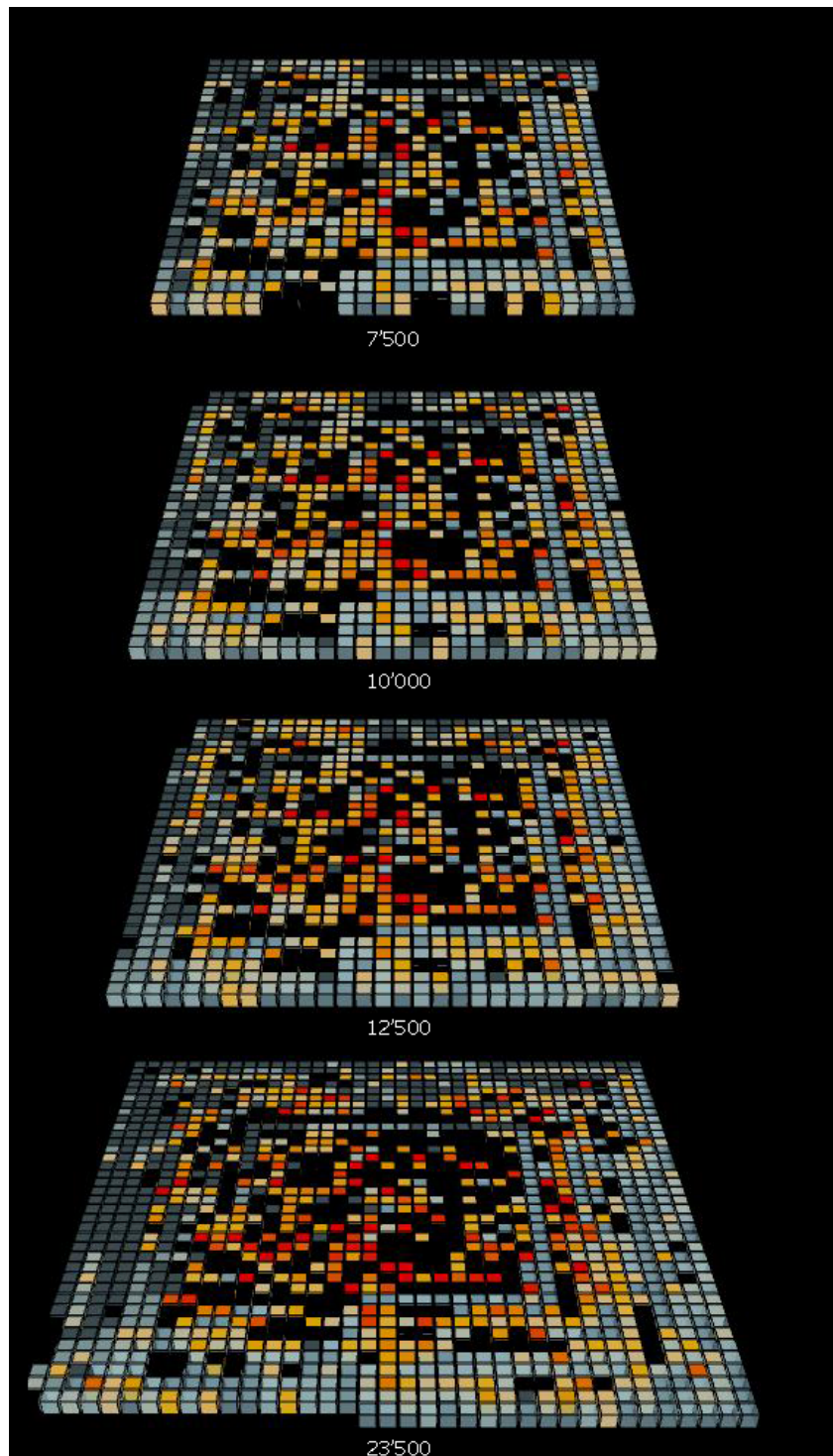


Figure 5.8: The frames from 7'500 to 23'500 of the epiphany commit history

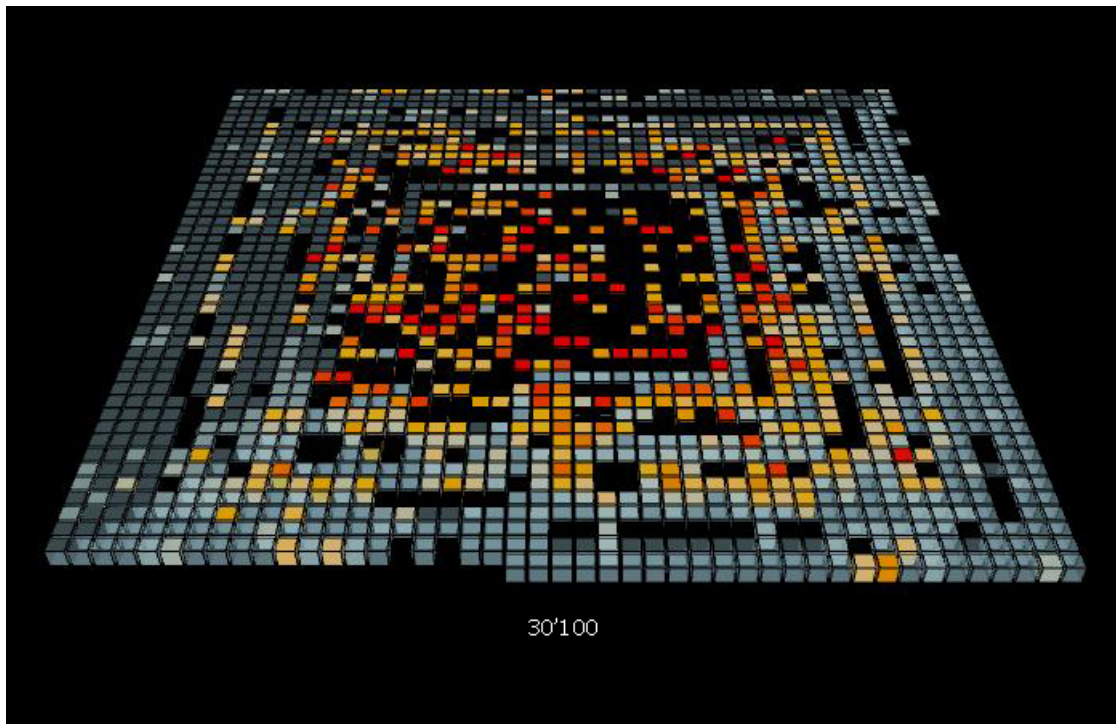


Figure 5.9: Animation frames at 30'100 of epiphany commit history

### 5.1.3 Test Case 3: Totem

Totem is a media player for the GNOME environment which runs on GNU, Linux, Solaris, BSD and other Unix and Unix-like systems. It already included in GNOME, its repository had around than 30'000 entries.

As we observe in fig. 5.10 the development of totem differs from the previous test cases, here we can see how there are significant changes to the system after few additions, we can see that at frame 100 the system already has been reshaped, files have been deleted, through frames 200 up to 500 the core functionality start to appear, the system has not grown very much yet, but significant changes have been made to certain files (colored in red- orange).

As the system grows (see fig. 5.11) it starts to show the same pattern as the previous test cases, large amounts of empty spaces that indicates that a pruning of files took place, also the core (red colors scattered around the center) and infrastructure files( orange-yellow colors enclosing the core files) are well identifiable. Moreover a lot of almost untouched files has been added, these files might be a series of files referring to additional plug-ins of the system.

We can notice how the Pareto principle applies as well in this case, the system changed significantly, but the core functionalities that could be identified at early stages of development are still present.

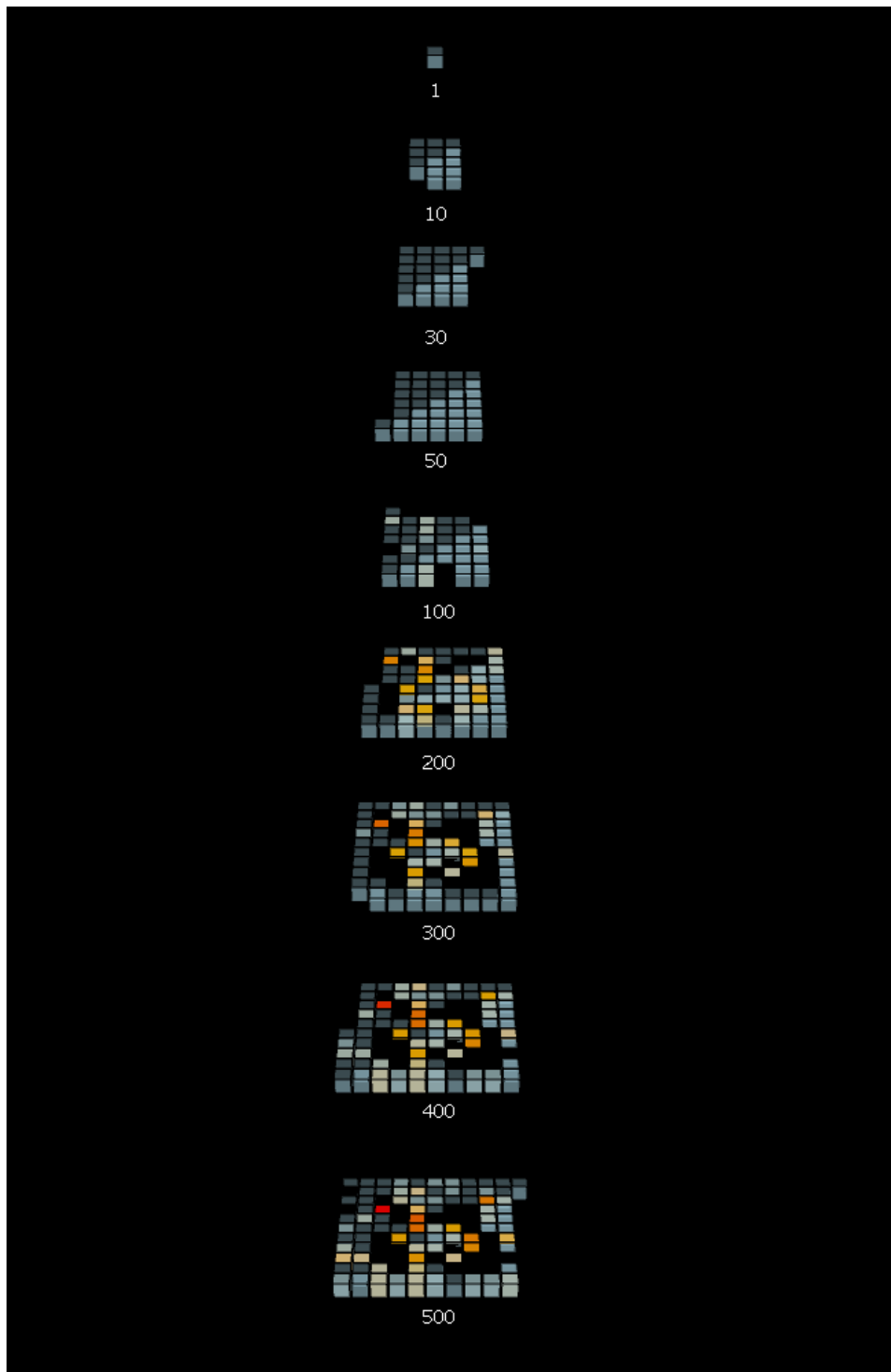


Figure 5.10: The first 500 frames of the totem commit history

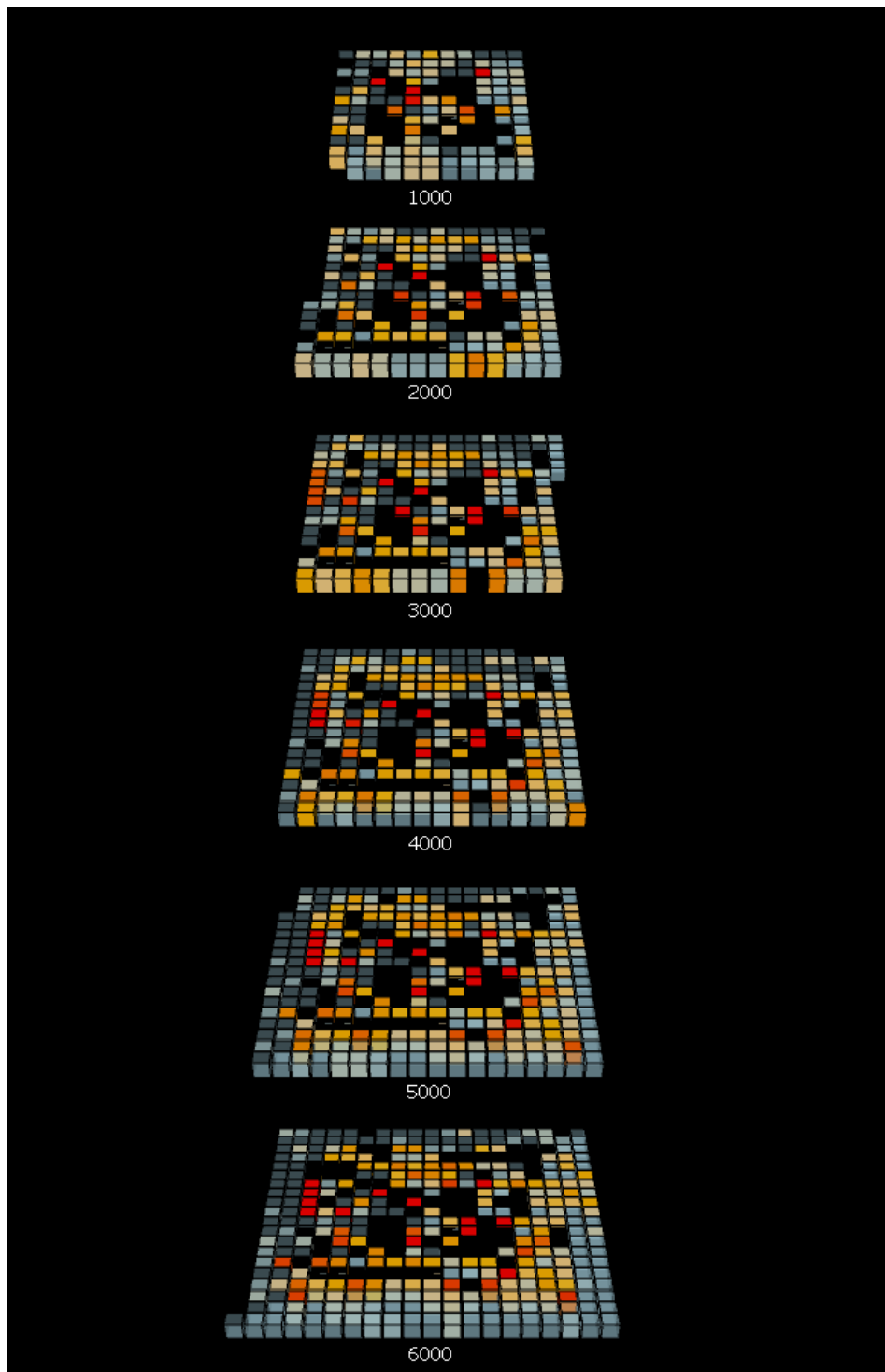


Figure 5.11: Totem commit history frames 1000-6000



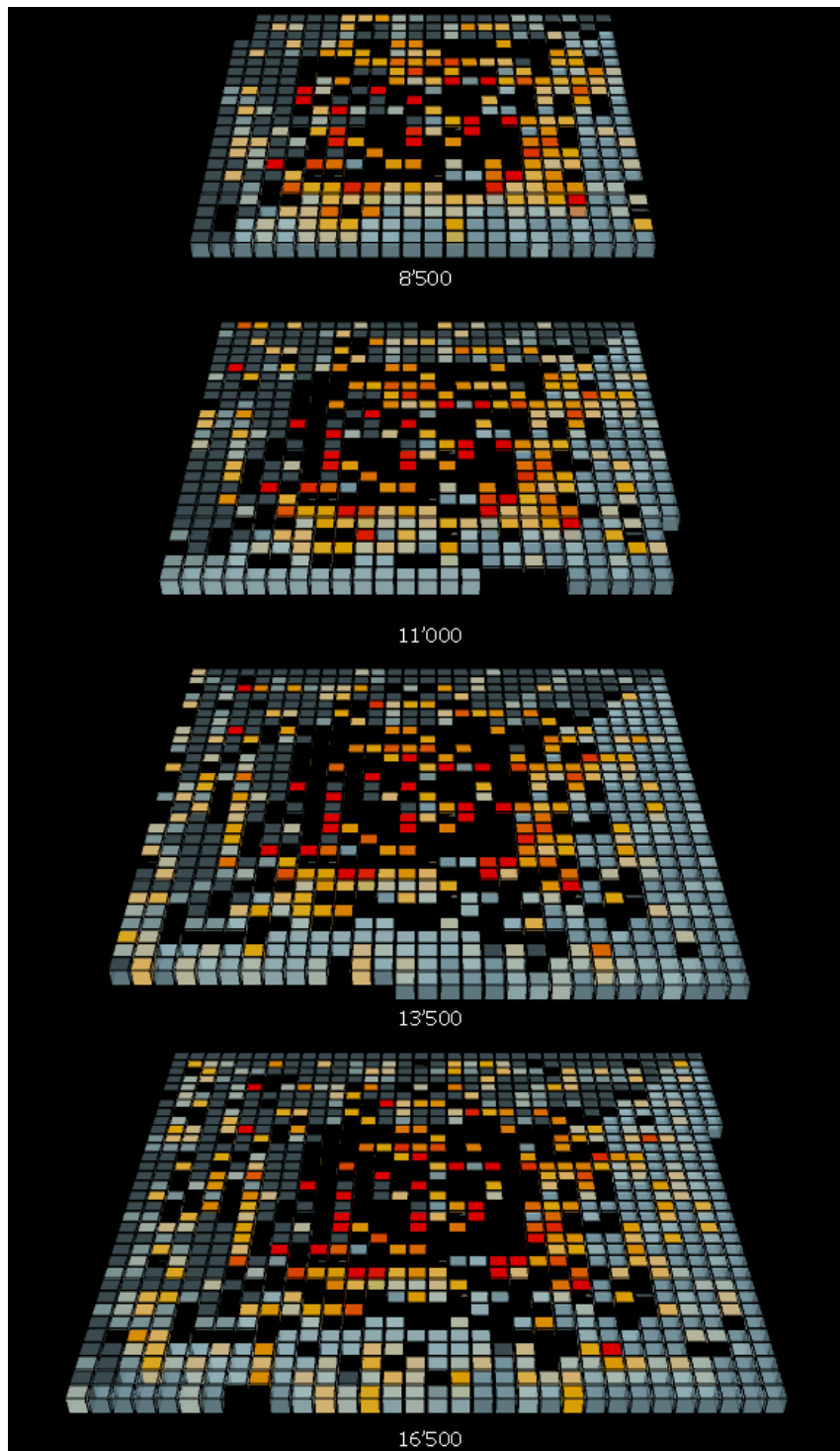


Figure 5.12: Totem commit history frames 8000-16500

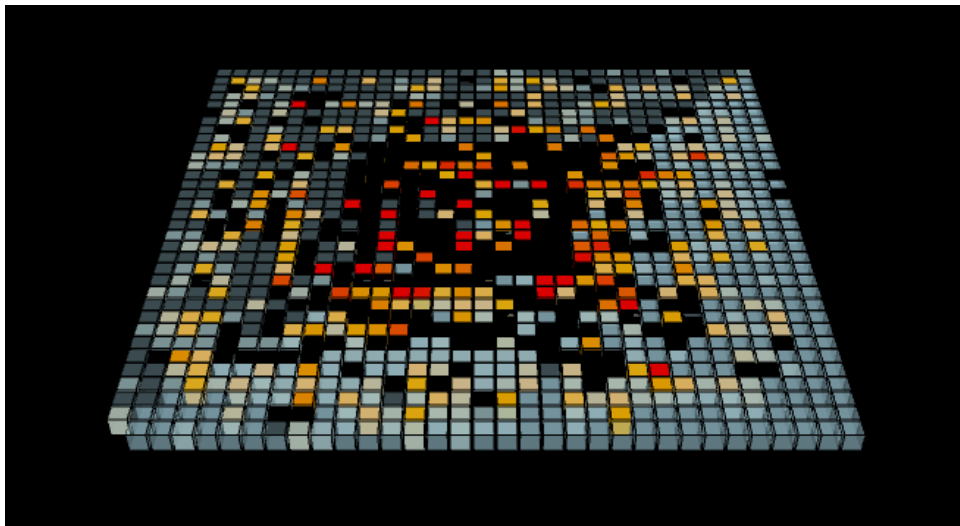


Figure 5.13: Totem commit history frames 17100



## Chapter 6

# Conclusion

As shown in the previous chapter, Lateralus is able to produce a visualization feedback of a system's evolution, moreover it allows to perform a visual analysis of the system at-hand and helps in further analytical reasoning. During the development of this tool I had the chance to gain insight in the field of software visualization. With Lateralus I think I met the goal of producing a tool that helps to understand the development process of software systems in the context of the evolution of software.

Lateralus is able to produce meaningful visualization about a system focusing on the files composing it, also in Lateralus the attention is to visualize the modifications that these files undergo during the development process. Moreover with Lateralus it is possible to track the evolution of a systems in terms of added and discarded files, making visible how much a system gets changed from its original state to the final release.

This experience was useful because it gave me the chance to understand the principles that are considered to be the theoretical foundations that are required to produce a meaningful visualization tool.

### 6.1 Future Work

In Lateralus there are basic functionalities that can be improved, the absence of a proper user interface makes this tool not yet ready to be made available for download. Moreover after some results I think that some old functionalities need to be reconsidered, in particular the heat map needs to be reviewed to mimic better the heat differences. A big improvement would be to have a screenshot manager that would be able to produce a stand-alone movie from the gathered screenshots. The main future goal would be to produce an intuitive User Interface for this tool in order to make it more usable. Also I'd like to fully integrate the Anar+ library kit, the experimental 3D view showed very interesting results, and might be an excellent starting point for a review of the tool. The intention is to complete Lateralus with a set of functionalities that would help gather additional information about files and authors.

# Bibliography

- [1] <http://vis.cs.ucdavis.edu/~ogawa/codeswarm/>. website.
- [2] <http://www.processing.org>. website.
- [3] U. Bern. <http://moose.unibe.ch/tools/chronia>. website.
- [4] K. S. B. Christopher G. Healey and J. T. Enns. *Acm. High-Speed Visual Estimation Using Preattentive Processing*, 1996.
- [5] M. D'Ambros. <http://www.inf.unisi.ch/phd/dambros/>. website.
- [6] M. D'Ambros and M. Lanza. *Journal of software maintenance and evolution: Research and practice. Visual Software Evolution Reconstruction*, 2007.
- [7] S. Diehl. *Visualizing the Structure, Behaviour and Evolution of Software*, 2007.
- [8] S. Few. *Tapping the Power of Visual Perception*, 2004.
- [9] M. B. H., D. T. A., and M. D. Brown. *Visualization in scientific computing. Computer Graphics*, 21, 6, 1987, 1987.
- [10] LaBelle and Nembrini. <http://anar.ch>. website.
- [11] P. M. Lanza. <http://www.inf.unisi.ch/faculty/lanza/>. website.
- [12] J. Malnati. *X-Ray, An Eclipse Plugin for Software Visualization*, 2007.
- [13] R. Robbes and M. Lanza. *Sciencedirect. A Change-based Approach to Software Evolution*, 2007.
- [14] R. Wettel. <http://www.inf.unisi.ch/phd/wettel/>. website.