INTERACTIVE AND COOPERATIVE VISUAL DATA MINING OF EVOLVING SOFTWARE

Master Thesis in Algorithmic and Information Modeling.

Thèse de Master, Algorithmique et Modélisation de l'Information.

CÉDRIC MESNAGE

Département d'informatique

Université de **Caen, France**

Under the direction of

PROF. MICHELE LANZA

Faculty of Informatics

University of **Lugano, Switzerland**

September, 2005

”If the doors of perception were cleansed, every thing would appear to man as it is, infinite.”

William Blake

Acknowledgments

- First of all I would like to thank my adviser, Michele Lanza, who gave me the opportunity to perform this work at the Faculty of Informatics of Lugano. He offered me both freedom and strong guidelines on my work.
- I thank all the members of the Faculty of Informatics of Lugano.
- My teachers from the University of Caen where I did my studies.
- My girlfriend, Elodie, who followed me to Switzerland and who supported me all that time.
- My parents, who have always given me the best and whom I love.
- All my friends in France and elsewhere, they are too many to cite them here.

Abstract

The maintenance, reegineering, and evolution of software systems has become a vital matter in today's software industry. With time systems tend to decay in quality. Such legacy software systems need to be reverse engineered in order to keep their value. One important source of information that is seldom used is the history of the system, i.e., the life-cycle it went through until its current state.

In this master thesis we propose a **visual data mining approach** to **reverse engineer** systems using the history data provided by **versioning systems**. We extract **software entities** from CVS (Concurrent Versioning System) and maintain these in a **data warehouse**.

Our visualizations are based on the **polymetric views** principle that we extend in **3D** and enrich with **data mining** techniques.

We implemented these visualizations in WHITECOATS , an interactive and cooperative tool working as a web-service, which we use to validate our approach in a case-study.

Contents

Chapter 1

Introduction

Reverse engineering existing software has become an important problem that needs to be tackled. Software systems evolve. Therefore, software system analyses must consider software evolution. New requirements, bug fixes and refactorings are part of a software's history. New developers and managers examine this history to understand the current state of a system. From the point of view of the software manager, introducing new functionalities in an existing software raise many questions such as ”How much time will it take for the system to be stable again?” or ”Who should I contact to do that?”.

By studying the software evolution, we are able to answer clues on these questions. Indeed if a similar functionality has been introduced before, analyzing the past of the system tells us what happened after this modification and who did it. Legacy software systems have years of history and consequently manually analyze these becomes impossible. Visualization techniques eases the understanding of such history.

1.1 Scope

Our work extends over many research areas.

Software Visualization is defined as ”the use of crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software. [16]”

Information Visualization is defined as ”the use of computer-supported, interactive, visual representation of abstract data to amplify cognition. [2]” The goal of information visualization is to visualize *any* kind of data.

Data Mining has been defined as ”The nontrivial extraction of implicit, previously unknown, and potentially useful information from data” [7] and ”The science of extracting useful information from large data sets or databases” [4].

Visual Data Mining is the use of visualization to understand large amounts of data and extract useful and new information. Visual data mining is used to elaborate classifications of data or to find outliers.

We define a model to visualize software information. These visualizations help visual data mining of software entities. Our model is generic and may be applied on other kinds of data. We offer some improvements on our visualizations using data mining techniques.

1.2 Thesis Outline

This document is organized as follow:

- In Chapter 2 we introduce the software evolution research field and describe our approach.
- In Chapter 3 we describe the principles of the WHITECOATS tool [13] and its user interface.
- In Chapter 4 we present the CVS module of WHITECOATS which is able to retrieve data from a CVS repository, structures and computes metrics.
- In Chapter 5 we explain how we use data mining techniques to facilitate visual data mining by arranging the subsets threads.
- In Chapter 6 we present how to make use of WHITECOATS visualizations to mine large sets of data.
- In Chapter 7 we present a case study based on the well-known open-source software, Azureus.
- In Chapter 8 we conclude by giving an outlook on future work.

1.3 A Short State of the Art

Software evolution has established itself as a research field in the past few years, leading to many publications in this area.

Jazayeri *et al.* analyzed the stability of the architecture [9] by using colors to depict the changes.

Taylor and Munro [17] visualized CVS data with a technique called *revision towers*. Ball and Eick [1] developed visualizations for showing changes that appear in the source code.

Gulla [8] proposes multiple visualizations of C code, but to our knowledge there was no implementation. Collberg *et al.* used graph-based visualizations to display the changes authors make to class hierarchies. However, they do not give any representation of the dimension of the effort and of the removals of entities.

Rysselberghe and Demeyer [15] propose a simple visualization of a system's history from a CVS repository by creating a matrix where the columns represent files ordered by names and lines represent the time. Wu, Holt and Hassan [18] introduce a spectrograph visualization to render software evolution. These approaches offer visualizations of software repositories, but no interaction with the visualizations are possible.

Lanza's Evolution Matrix [11] visualizes the system's history in a matrix in which each row is the history of a class. A cell in the Evolution Matrix represents a class and the dimensions of the cell are given by evolutionary measurements computed on subsequent versions. However, the Evolution Matrix has been implemented in a stand-alone tool (CodeCrawler) and uses only two-dimensional visualizations.

Chapter 2

Software Evolution

Versioning systems help developers to work together by storing their work at each step of the evolution of the software system. We recover software systems' history by extracting data from versioning systems. Using this history, we analyze and visualize the software evolution.

We see in this chapter which versioning systems are available and how we use it.

2.1 Versioning Systems

In [14], Robbes compares different kinds of versioning systems from the point of view of a software evolution researcher. File-based versioning systems such as CVS, SubVersion, and SourceSafe store revisions and changes on a file granularity. CVS has no refactoring support, such as renaming of files or function signature changes. Subversion has changeset support where the developer can set a collection of files. If files are in the same changeset, then changes on these files are associated to the changeset instead of the file. Changesets help refactoring support. Subversion versions directories as well. From a refactoring point of view, it detects if a file is moved or renamed.

Entity-based systems, such as StORE, version changes on a different granularity. Packages, modules, classes, and methods are versioned. It has refactoring support. The renaming of a class does not create a new entity, but instead adds a new event on the existing entity. The whole history of a class can be recovered even if it changed or moved several times in the history of the system.

2.2 The Problem

To analyze a software system, one possibility is to reverse engineer this system using the data provided by the versioning system.Versioning systems store the evolution of software systems, but to extract the history of the system we need to parse some data. CVS offers only log files which are text files containing a list of events. Reverse engineering is defined by Chikofsky and Cross as "the process of analyzing a subject system to identify the system's components and their relationships, and to create representations of the system in another form or at a higher level of abstraction [3]."

Most current reverse engineering tools are standalone programs. They ease the understanding of the structure and the evolution of a software system, but making a cooperative analysis is difficult. Different human resources around a project have distinct point of views on the same system. Therefore the analysis can not be done by only one person. Doing a collaborative analysis enhance the results.

Some tools are available on the Internet in the form of web-sites (ViewCVS [1], Maven [2]). They offer simples visualizations without interaction.

2.3 Our Approach

We propose a visual data mining approach based on *polymetric views* to tackle the problem of extracting information from software evolution data. The concept of polymetric views was introduced by Lanza in [12]. The main idea is to create **visual figures** from **software entities**. These figures are positioned and shaped according to the software metrics of the entities. In three dimensions we are able to display up to 7 metrics on the same figure to create visualizations of many entities.

Our software entities are extracted from reverse-engineering of CVS data. It is the most used versioning system. It gives us more projects to analyze. Our approach being based on modules, it is easy to adapt a new data source based on other versioning systems.

We introduce the concept of *subsets threads* which represents entities structurally related and sequentially ordered (*i.e.,* revisions of a file) on **polylines**. These polylines are sorted by similitude in order to visually compare them.

The previous concepts have been implemented in a tool, WHITECOATS , which displays visualizations within a web-site.

[1]ViewCVS is a browser interface for CVS and Subversion version control repositories, http://viewcvs.sourceforge.net/

[2]http://maven.apache.org/

Chapter 3

WhiteCoats, a Visualization Tool on the Web

We present in this chapter the main principles regarding visualization. WHITECOATS is the tool we developed to visualize software evolution data.

3.1 The Principles

We display both 3D and 2D visualizations. These are based on the concept of polymetric views [12]. A view is a combination of figures which is the result of a mapping of entities' metrics on the visual properties of the figures.

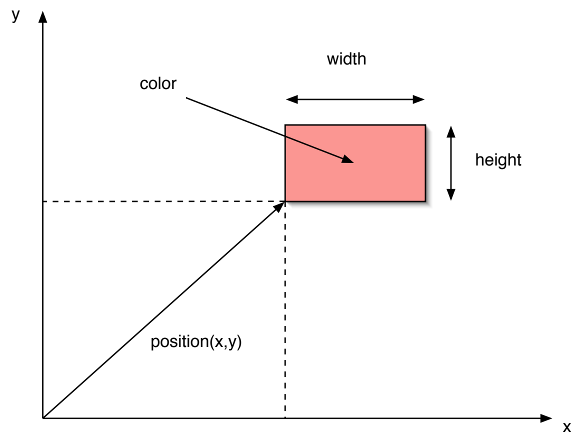


Figure 3.1. 2D Polymetric view principles.

The 2D polymetric view principle [12] permits visualization of large sets of entities. Figures' positions depends on their associated entities' metrics. Up to five visual properties can be mapped from metrics - the x and y position, the width and height, and the color (see Figure 3.1).

Extending the concept of polymetric views to 3D, the user can set a number of metrics which influence the position and shape of the visualized entities (see Figure 3.2). We map up to seven metrics (three for the position, three for the shape and one for the color). The point is not to display as much information as possible, but to discover views whose combinations of metrics convey certain types of information to the viewer.

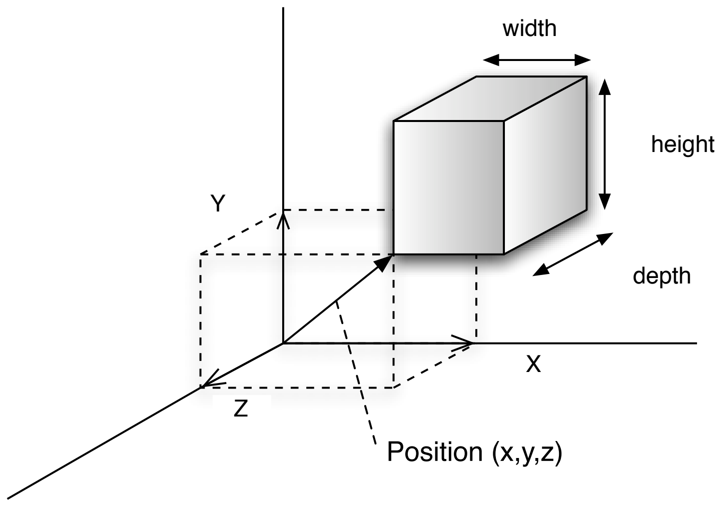


Figure 3.2. 3D Polymetric view principles.

In order to be transformed into figures, entities must follow the model presented as UML class diagram in Figure 3.3. An entity has an unbounded number of metrics which are defined by a name and a value. An entity has a type and a name. The entities are related by an association parent/children. The children of an entity are the entities related to it (*i.e.,* revisions and authors of a file).

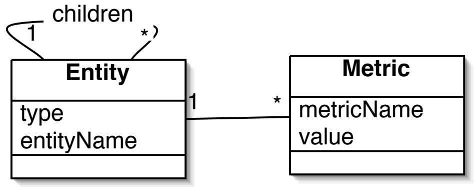


Figure 3.3. The data model of WhiteCoats.

This model may be generic enough to handle other kinds of data, but at the time of the writing of this document, we focus on software evolution information.

3.2 WHITECOATS Architecture

To display 3D visualizations in web-sites, we chose to use VRML[1]. VRML gives us the ability to easily create files defining the 3D scene and the plugin in the web-browser takes care of the rendering process. In the following, renderer mean creation of a file.

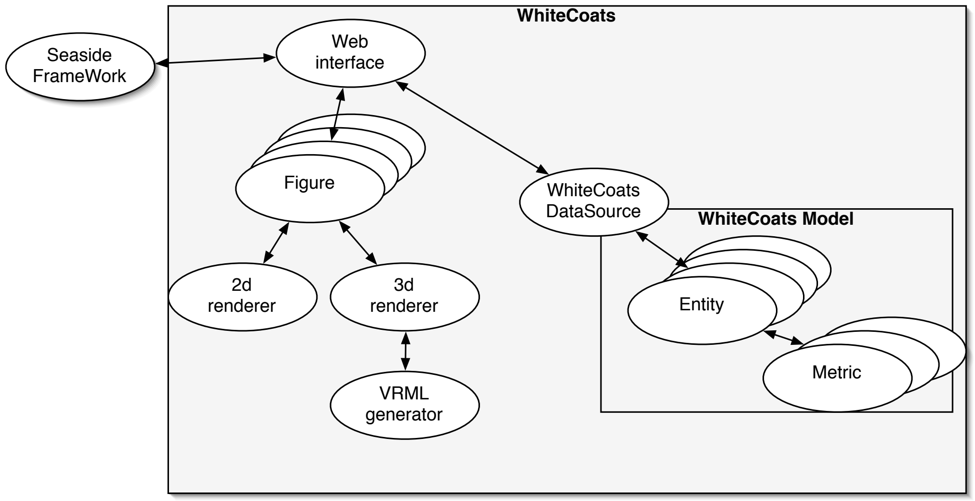


Figure 3.4. The overall architecture of WHITECOATS **.**

WHITECOATS is composed of different modules:

- The web user interface module which uses the Seaside framework [2] and gives the user the ability to visualize, browse or configure his visualizations.
- The data sources which contain the entities to be processed during visualizations and which will be transformed into figures after the configuration.
- The 2D and 3D renderer create visualizations from the figures.
- The VRML generator which is used by the 3D renderer to create VRML visualizations.

In Figure 3.4 we see the relationships between these modules.

[1]Virtual Reality Markup Language
[2]Seaside is a web-framework designed in smalltalk, http://seaside.st/

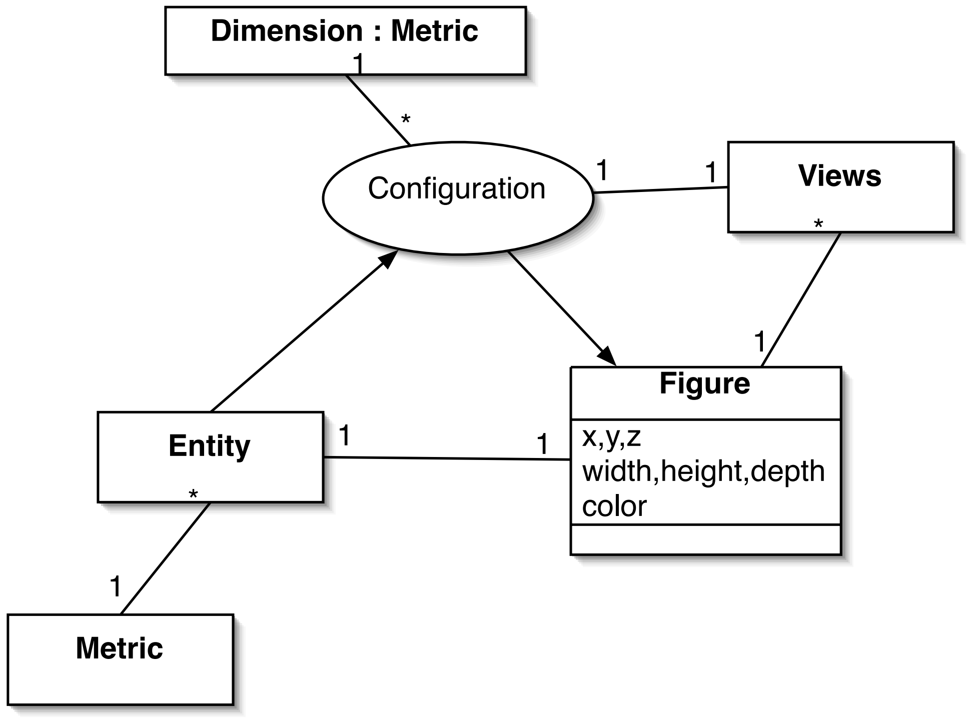


Figure 3.5. Creation of figures from entities.

The visualizations display figures. The visual properties of these figures are mapped from metrics. In Figure 3.5, we present the configuration process.

1. The user sets a configuration mapping software metrics to visual properties.
2. From the application of the configuration on the entities, we create visual figures.
3. The figures are rendered in a view (2D picture or 3D VRML file) which is conformant to the configuration.

3.3 A Collaborative Tool

Making our tool a web-application eases collaboration. Users are able to add comments on each entity. Seeing the comments of other people in interaction with the software system facilitates the process of understanding and may result in a production of more useful information. WHITECOATS can be seen as a web-forum in which the interaction and navigation are based on visualizations.

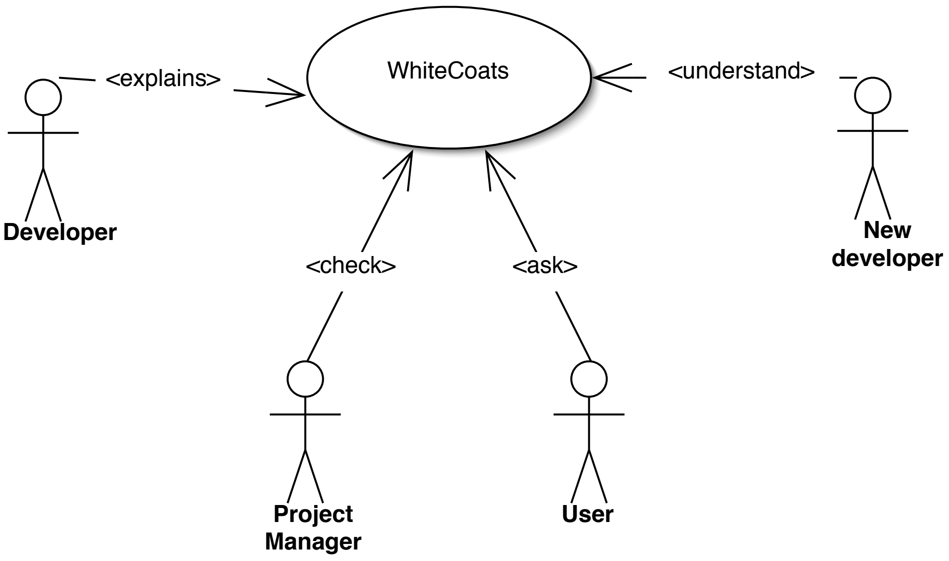
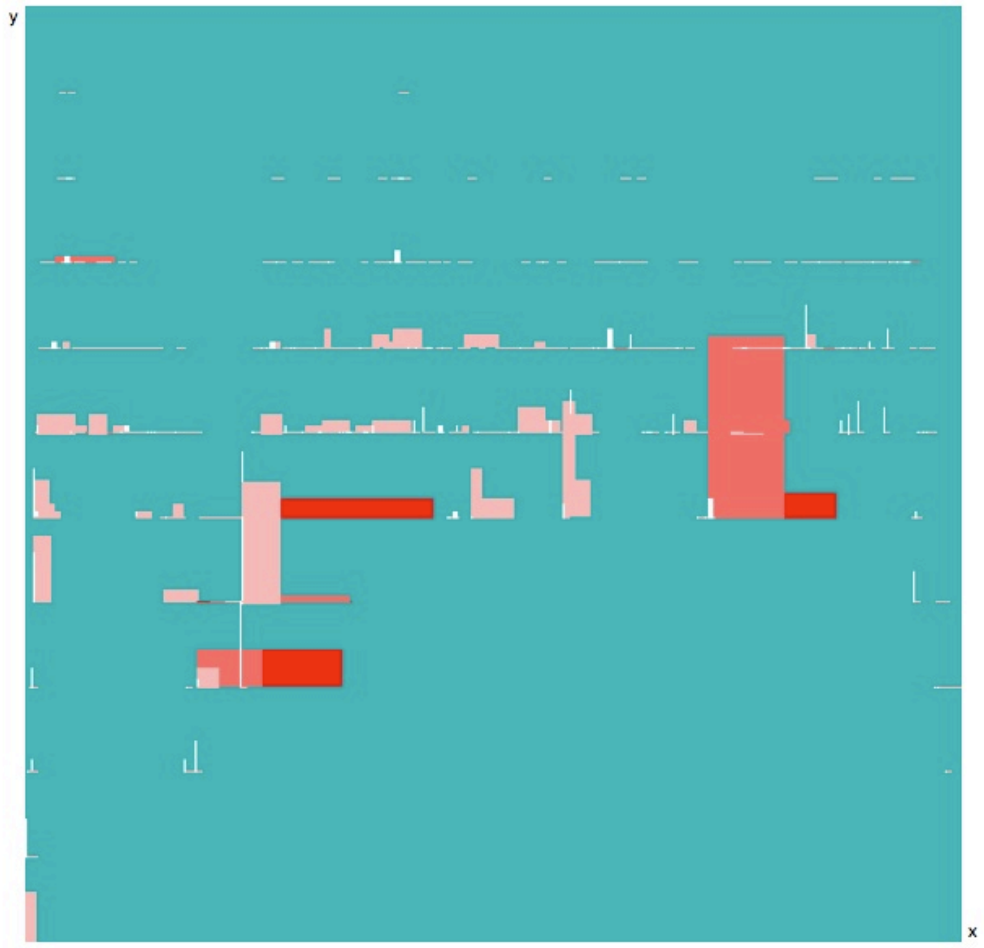


Figure 3.6. Different users and their relation to WHITECOATS **.**

Many different kinds of users (Figure 3.6) can use the same WHITECOATS application , such as the project manager, to the user of the software, the developer. The analysis comes from different points of view, and thus is enhanced.

3.4 Examples

The following selected examples come from Azureus [3], a bittorrent[4] client. Azureus was, at the time of writing of this thesis, the most active project on Sourceforge[5] . Sourceforge is also the most used repository for open-source software systems.



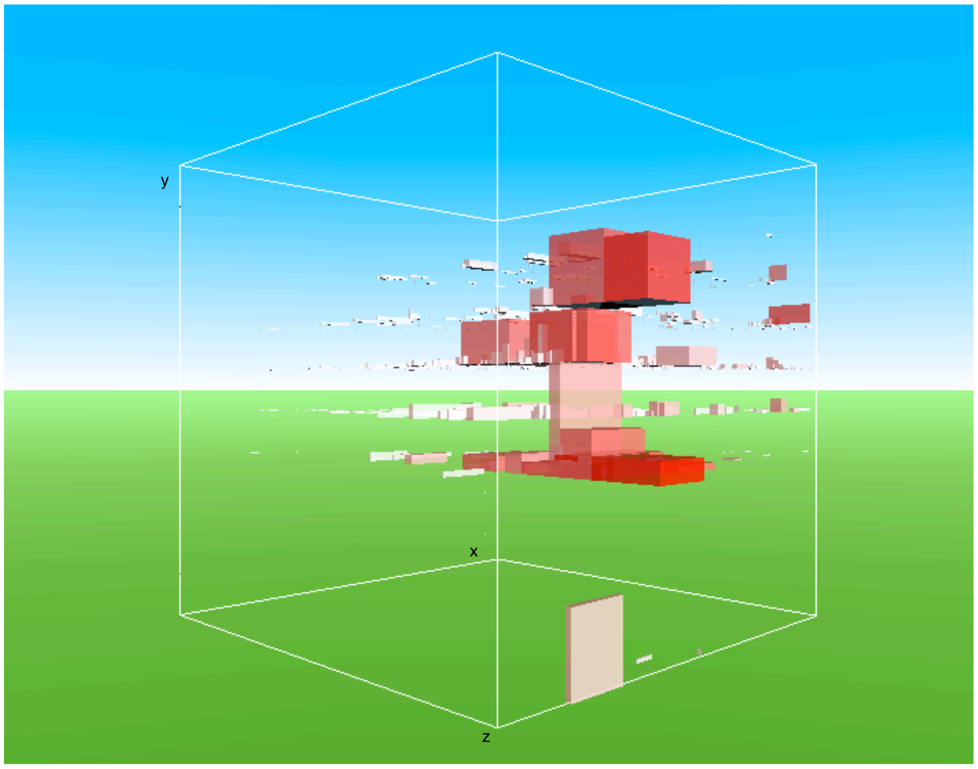
Entities	X	Y	Width	Height	Color
Directories	Depth	Parent ID	Number of files	Number of subdirectories	Number of files

Figure 3.7. A 2-dimensional polymetric view, Directories of Azureus.

2d Directories view. Figure 3.7 is useful to analyze the partition of directories in a system. In this view we compare directories according to their size (number of subdirectories,

[3]See http://azureus.sourceforge.net/
[4]http://bittorrent.org
[5]http://sourceforge.org

number of files), when we structure it using the depth metric (which is the depth in the directory hierarchy).



X	Y	Z	Width	Height	Depth	Color
First revision	Depth	Last revision	Number of authors	Number of revisions	Added lines	Removed lines

Figure 3.8. The Files Activity view of the Azureus project.

Files Activity view. In Figure 3.8, we show all the files of the Azureus project (over 3500 files). Using this view we compare files by their position on both evolution oriented metrics (First revision, Last revision, Number of revisions) and structural metrics (Depth, Added/Removed lines).

Chapter 4

Handling and Visualizing CVS Data

WHITECOATS is a visualization tool which can handle any kind of data, as long as it is conformant to our model previously defined. We developed a module to handle *software evolution data*. Currently this module extract its data from CVS repositories, but it can be easily adapted to other versioning systems. The architecture of this module can be reproduced to handle other versioning systems, but will need to be re-written as many other information can be available. The data warehouse manage a Postgres database, we describe here its populating processus.

4.1 WHITECOATS CVS Architecture

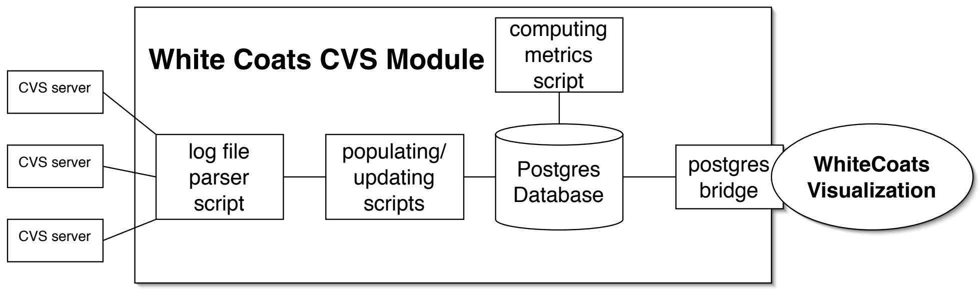


Figure 4.1. The overall Architecture of the White Coats CVS Module.

In Figure 4.1 we present the architecture of the module interacting with CVS. In order to use CVS data we need to fetch and then analyze them ; we store the results of the parsing in a database. This processus is composed of three main parts:

1. Retrieving/Parsing information from the CVS repository.

2. Populating or updating the Postgres database with the new data.
3. Computing metrics and associate them to entities in the database.

CVS stores a revision each time a file is changed. When retrieving the log informations we get information on each revision. From all these informations we populate a Postgres[1] database using a set of scripts written in python[2]. These scripts take care of fetching new data from the CVS repository and parsing log files to extract entities from it. Other scripts take care of populating the database and computing metrics from these newly created entities.

The postgres Bridge is an interface between the WHITECOATS visualization engine and the database. It creates entities conformant to the WHITECOATS model from the entities in the data warehouse.

4.2 The Data Warehouse

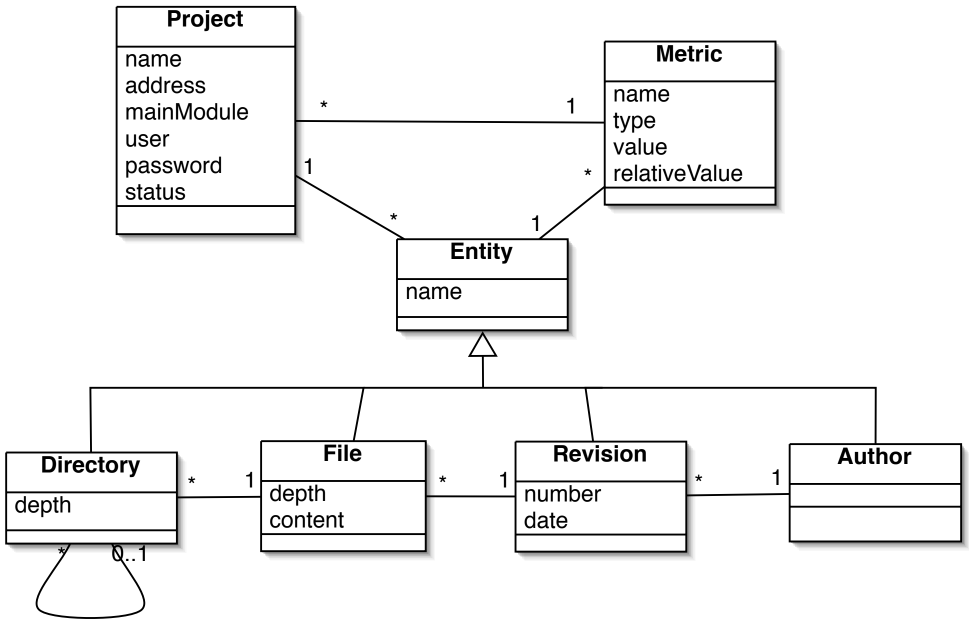


Figure 4.2. The WHITECOATS Database model.

The database model (Figure 4.2) is inspired from the RHDB (Release History Database) [6] [5]. The main difference between the RHDB and our model is that we store metrics in the database.

Some of the metrics (see Table 4.1) are computed from SQL queries . WHITECOATS allows metrics to be pre-computed or computed on the fly, but considering the time to compute

[1]http://www.postgresql.org/
[2]http://www.python.org

metrics of tens of thousands of entities, the resulting latency would decrease the usability of WhiteCoats.

Directories, files, revisions and authors are subclasses of Entity. Entities have metrics. The database can be seen as a tree. The root is a project which corresponds to the CVS repository. A project contains directories which are arranged in a tree. Directories can contain files. CVS revisions are stored as an association between a file and an author.

This model facilitates the recovery of other informations that are not extracted from CVS such as directory versioning or author history. The data warehouse is composed of the database and the scripts to manage it. The addition of a new project in the data warehouse is done in three steps. We describe them below:

1. **Populating the Database.**

   To process evolution information, we use a set of scripts to populate a database which is then the source of information for the visualizations. The populating process (Figure 4.1) is done as follows:

   (a) The user indicates the URL of a CVS repository.

   (b) A set of scripts is executed that connect to the repository and download all log files which contain information about every revision of every file. The log files are parsed and a database is populated with the retrieved information (see Figure 4.2 for the DB schema we use). A set of directly retrievable metrics (*e.g.,* number of added lines) and other information (*e.g.,* id of the author) is also stored into the DB.

   (c) A second set of scripts computes information that cannot be retrieved directly (see Table 4.1) and stores them into the DB too.

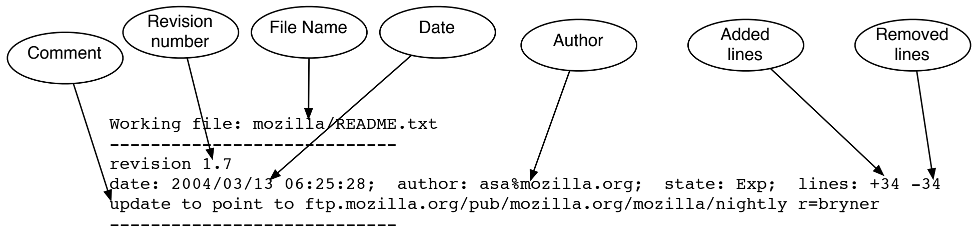
```
=============================================================================

RCS file: /cvsroot/mozilla/README.txt,v
Working file: mozilla/README.txt
head: 1.7
branch:
locks: strict
access list:
symbolic names:
        MOZILLA_1_7_11_RELEASE: 1.7
        ...
        MOZILLA_0_9_9_BRANCH: 1.1.0.2
keyword substitution: kv
total revisions: 12;    selected revisions: 12
description:
----------------------------
revision 1.7
date: 2004/03/13 06:25:28;  author: asa%mozilla.org;  state: Exp;  lines: +34 -34
update to point to ftp.mozilla.org/pub/mozilla.org/mozilla/nightly r=bryner
----------------------------
revision 1.6
date: 2003/09/18 04:31:38;  author: imajes%php.net;  state: Exp;  lines: +0 -1
updating
```

Figure 4.3. Extract of a CVS log file of Mozilla.

We store in the database the date when the checkout of the CVS project has been done last time. Using this date, we can get the CVS log file for only the revisions which appeared between that date and now. Our update script finds entities which have already been inserted in the database, updates their data, and creates new entities for files, directories, and authors.

2. **Parsing the Log File.**



```
Working file: mozilla/README.txt
----------------------------
revision 1.7
date: 2004/03/13 06:25:28;  author: asa%mozilla.org;  state: Exp;  lines: +34 -34
update to point to ftp.mozilla.org/pub/mozilla.org/mozilla/nightly r=bryner
----------------------------
```

Figure 4.4. Information that can be retrieved from a CVS log.

A CVS log file is a text file (see Figure 4.3) which contains informations on revisions of each file. It contains as well informations on branches and releases, but we do not use them. We present in Figure 4.4 a revision in a log file and all information that we retrieve from it.

From the file name we create many entities, directories of the path of the file, and the file itself (*i.e.,* "mozilla/src/main.c" will result in two directories, "mozilla" and "mozilla/src", and a file "mozilla/src/main.c"). For each revision of the file we create a revision entity with a time and date stamp. We set its number of added/removed lines. The author is created too from his name if he is not already present and the revision is associated to this author. The comment is stored in the revision entity. The date and added/removed lines will later be used as metrics.

Once the parsing is done, we look in the database for entities already inserted for that project. Then the entities which are not already in the database are inserted and linked to the old or new ones if needed (*i.e.,* a directory of a new file may already exists, so we look in the database for it).

When all data are inserted, we can compute the metrics for each entity.

3. **Computing Metrics.**

Some metrics are retrieved from the CVS logs (such as the number added or removed lines), while others are computed in a second phase. In Table 4.1, we list some of the metrics. The metrics are computed in a preprocessing step.

SQL Queries like the ones in Figure 4.6 and Figure 4.5 are executed by a python script in the Postgres database.

In Figure 4.5 the query creates new metrics for directories setting the name of the metric with 'Number of files in directory' and the value to the number of files with the same directory id. This is done by the use of a count(file.id) and a Group By clause.

Property Name	**Entity types**
Retrieved from CVS Log Files	
Added/Removed lines	Revision
Author	Revision
File	Revision
Revision Date	Revision
Simultaneous Revisions	Revision
Computed	
Depth	File, Directory
First/Last Revision	File, Directory, Author
Added/Removed lines	File, Author
Active period	File, Author
Number of Authors	File, Directory
Number of Revisions	File, Directory, Author
Number of Files	Directory, Author
Parent Directory	Directory

Table 4.1. Entity Properties.

```
""" INSERT INTO Metric (project_id, entity_directory_id, type,  name, value)
SELECT %s, directory_id, 'nofD' ,'Number of files in directory' , count(file.id)
from file where file.project_id=%s GROUP BY directory_id;
"""%(project_id, project_id)
```

Figure 4.5. SQL query to compute and insert the number of files within each directory.

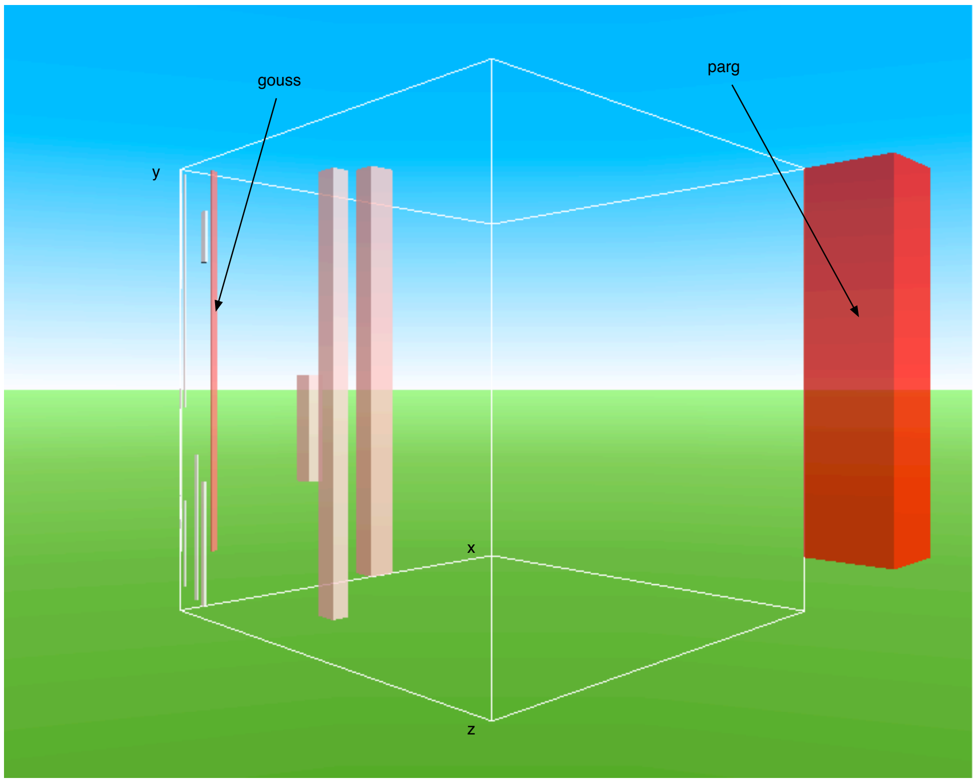
In Figure 4.6 the query computes the total of added lines by each author and creates a new metric assigned to each author and named 'Added lines'.

```
""" INSERT INTO Metric (project_id, entity_author_id, type,  name, value)
select %s, author.id, 'aLA', 'Added lines', sum(metric.value)
from author,revision, metric
where revision.project_id = author.project_id and author.project_id = %s
and author.id=revision.author_id and revision.id=metric.entity_revision_id
and metric.name='Added lines' GROUP BY author.id;"""%(project_id,project_id)
```

Figure 4.6. SQL query to compute and insert the total number of added lines for each author.

4.3 Examples

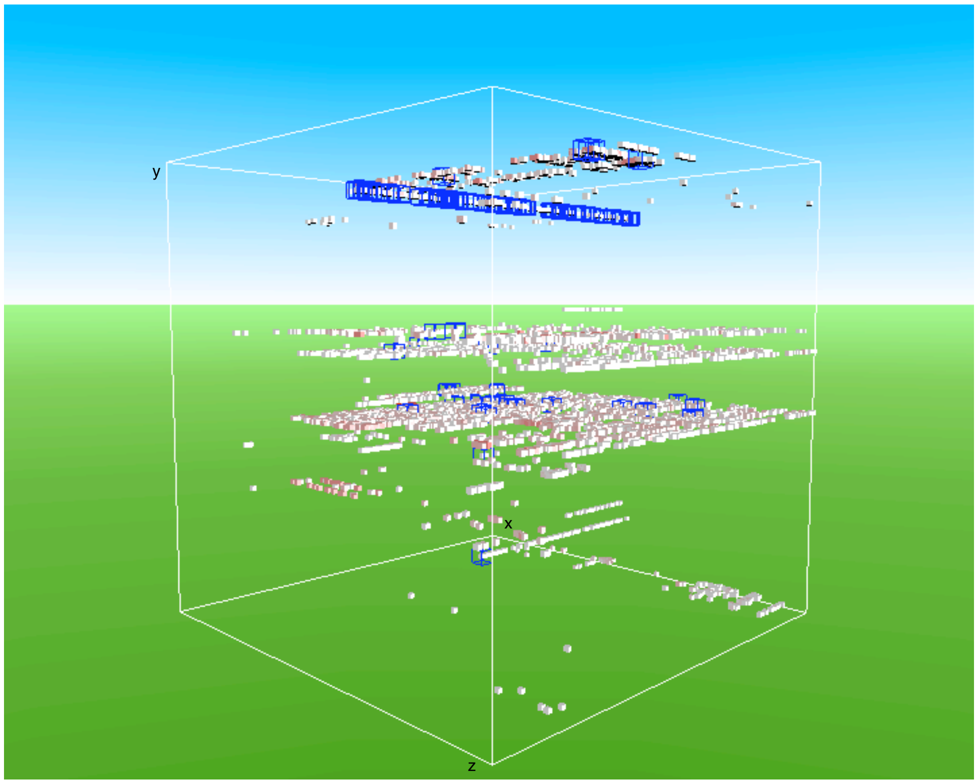
Using the data extracted with our module in WHITECOATS , we are able to create new visualizations of evolution data.



X	Y	Z	Width	Height	Depth	Color
Number of revision	First revision	Number of files	Number of revisions	Activity Period	Number of files	Added lines

Figure 4.7. The Author's activity of the Azureus project.

Authors view. In Figure 4.7 The authors are positioned and shaped according to their importance in the project (Number of files, Number of revision, Added lines). The Y axis represent the date of the first revision, which is the entry of the author in the project, and the height attribute is mapped to the metric activity period, which is the difference between the date of the last revision and the date of the first revision. This view tells us who is working on the project, when they started and how much did their work last.



Entities	X	Y	Z	Color
Revisions	Date	Author ID	File ID	Number of revisions at the same time

Figure 4.8. The 3D Evolution matrix of the Azureus project.

3D Evolution Matrix view. In Figure 4.8 we show the Evolution Matrix ([10]) of the project Azureus which represents over 15'000 revisions. We added a query which highlights in blue the revisions of more than 300 added lines. This example shows also the limits of the paper media to explain the possibilities of White Coats: the user would now navigate ("fly through" or "rotate") to this file history and have a closer look.

Chapter 5

Improving Visualization with Data Mining

Data mining has been defined as ”The nontrivial extraction of implicit, previously unknown, and potentially useful information from data” [7] and ”The science of extracting useful information from large data sets or databases” [4].

In the previous chapter we have seen what kind of visualizations we can do to visualize evolution data (Figure 4.8). In the evolution matrix, each cube is a revision which happened on a file. When the number of entities displayed is large (over 16’000 in Figure 4.8) it becomes difficult to differentiate each file’s revisions.

In order to improve these visualizations, we introduce the concept of **subset threads**. Instead of having many entities in a visualization, we have polylines joining children of the same product (in our case joining every revision of the same file). This way instead of having over 16’000 cubes, we get 3’000 polylines. In addition to the *subset threads*, we propose an algorithm to sort these threads by **similitude**. Once sorted the threads can be compared by the user.

5.1 Definitions

In this section we define *subset threads*. A *subset thread* is what we represent as a polyline in visualizations, it is a set of children of an entity of the same type (*i.e.,* revisions of a file). Each entity of a subset corresponds to one point of the polyline according to the metrics values of the entity (*i.e.,* in the evolution threads, the x axis is the date of the revision, the y axis the number of added lines).

- A subset of an entity e of type t is a set of entities $e_1..e_n$ of type $t2$ where for $0 < i <= n$ the entity of type t related to e_i is e.

- A figure of an entity e is, given a metric mx and a metric my, the point (x, y) where $x =$ the metric mx of e and $y =$ the metric my of e.

- A subset thread of an entity e is, given is subset $e_1..e_n$, the polyline formed by the figures $(x_1, y_1)..(x_n, y_n)$ where $x_1 < x_2 < .. < x_n$.

5.2 Correlations between Subsets

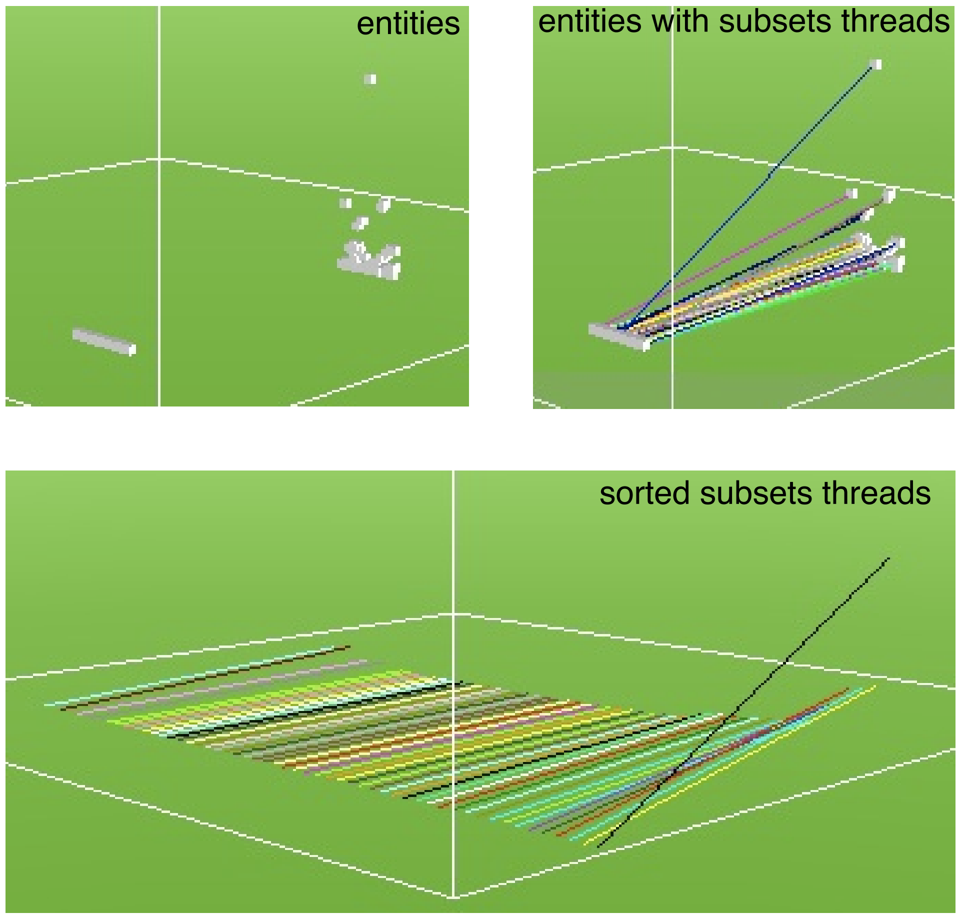


Figure 5.1. An example of sorted subsets threads.

In Figure 5.1 the entities displayed are revisions, the x axis representing the date of the revision and the y axis the number of added lines. The z axis is associated to the file Id in the top figures.

From the entities displayed in the top left in Figure 5.1, we can extract many subsets as defined in the previous section each corresponding to the revisions of a file. If we display the polylines corresponding to the subsets, we obtain the top right figure, each entity in the same subset are connected with a line (each revision on the same file).

The subsets help the visualization of large sets of entities. In the top left figure, it is hard to find revisions of the same file, even if they are on the same z axis. In the top right figure, we just have to follow a line to find the next revision of a file.

The bottom screen capture represents the same subsets but the polylines are sorted by similitude and the entities are not displayed. We can compare the lines and find files which evolved the same way.

5.3 Computing Similarity

The aim is to sort a set of *subsets threads* according to their correlation. Given a first line, we look for the line which is the more similar to this one and position it, then we do the same process for the new line.

When comparing two polylines, we combine the two lines and then compute the perimeter of the new line formed by the two lines. This gives us the possibility to sort the subsets and then positioned them according to their similitude.

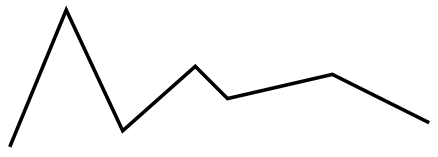


Figure 5.2. The first polyline.

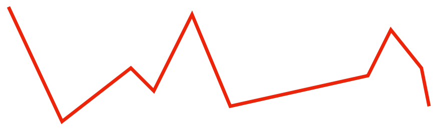


Figure 5.3. A polyline to compare with the first one.

Let a and b be two polylines (Figure 5.2 and Figure 5.3) composed by points $a_1..a_n$ and $b_1..b_m$.

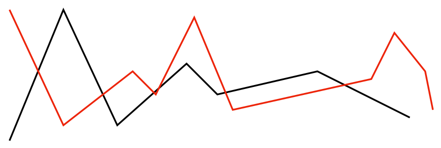


Figure 5.4. The two polylines.

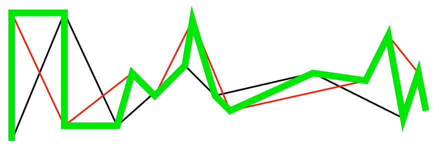


Figure 5.5. The polyline formed by the combination of the two polylines (in green).

The composed polyline c (in green in Figure 5.5) of points $c_1..c_{m+n} \in a_1..a_n \cup b_1..b_m$ and $x_1 < x_2 < .. < x_{m+n}$.

The perimeter of the polyline c is smaller when the two lines are close to each other. We use this metric to sort the *subsets threads* as we see in the next section.

5.4 Algorithms and Complexity

In the algorithm 1, let S be the size of $subsets$ then the complexity of the loop is in $O(S)$, lines 2 and 3 are constant. To find the complexity of line 4, we must analyze the algorithm 2.

In the algorithm 2, if we assume that the computation of the perimeter in line 6 is in $O(N)$ where N is the number of points of a line, and the main loop (line 3) is in $O(S)$ then the complexity of the algorithm 2 is in $O(S * N)$.

The complexity of the algorithm 1 is then in $O(S^2 * N)$. Considering that the number of subsets is generally small, this algorithm is efficient.

Algorithm 1 Sort subsets threads Algorithm.

- Entry : l, the first line to be insert and $subsets$, the list of polylines.
- Result : $sortedsubsets$, the lines sorted and positioned

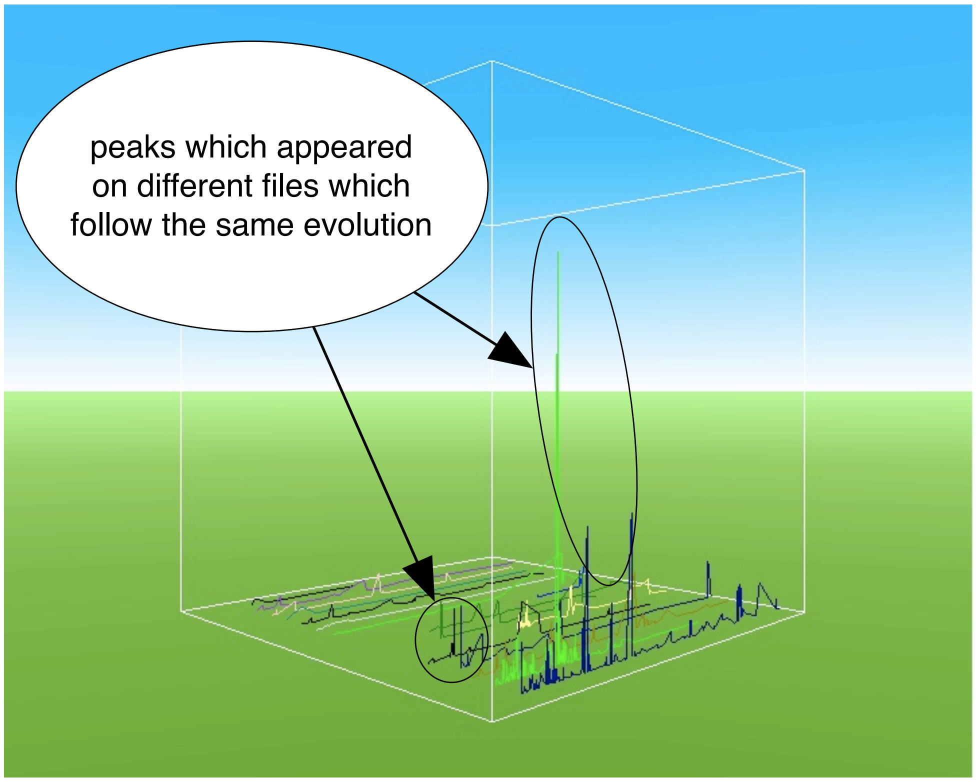
```
1: while $subsets$ not empty do
2:     remove $l$ from $subsets$
3:     add $l$ to $sortedsubsets$
4:     $l \leftarrow$ most similar line to $l$ within $subsets$
5: end while
6: Return $sortedsubsets$
```

Algorithm 2 Most similar thread Algorithm.

- Entry : l, a polyline and $subsets$, the list of polylines not yet sorted.
- Result : the closest line to l and a distance.

```
1: $minp \leftarrow bigint$
2: $closest \leftarrow l$
3: for all $l2 \in subsets$ do
4:     combine $l$ and $l2$ in $l3$
5:     sort the points of $l3$ on the $x$ axis
6:     $p \leftarrow perimeterofl3$
7:     if $p < minp$ then
8:         $closest \leftarrow l3$
9:         $minp \leftarrow p$
10:    end if
11: end for
12: set $z$ coordinate of $closest$ to $minp$
13: Return $res$
```

5.5 Example


peaks which appeared on different files which follow the same evolution

Entities	X	Y	Color
Files	Date of revision	Added lines	Random

Figure 5.6. Evolution Subset Threads.

In Figure 5.6, we present the **Evolution Subset Threads** view. This view show sequential data of different products (in our example files). The threads are sorted by similitude using the algorithm presented in this chapter.

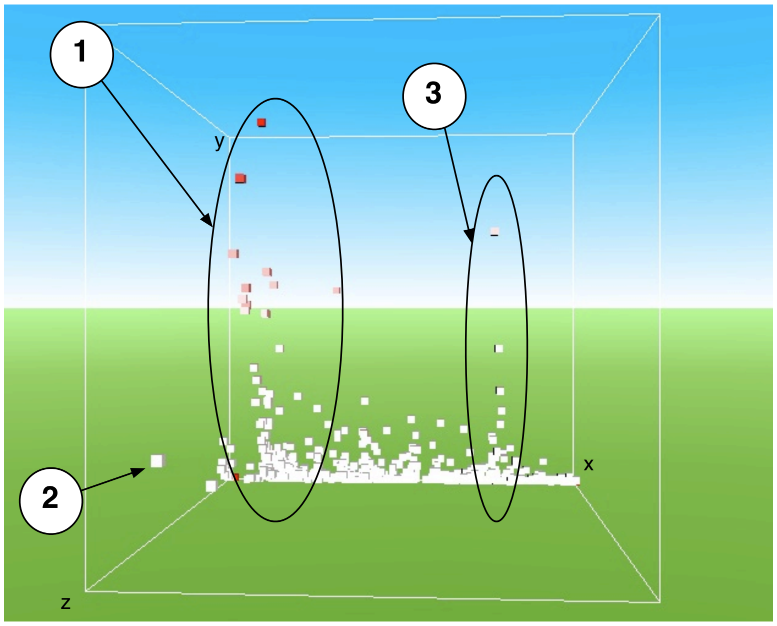
Comparing the evolution of files is a problem that still need to be tackled. Our visualization eases the visual comparison for we position similar evolutions next to each other. We are now able to describe groups of similar evolution. Using the color to map another metric gives the ability to detect flouted evolutions (files which evolves with files reposing in another directory, see next chapter).

Chapter 6

Visual Data Mining

We have seen in the previous chapters how we create visualizations in WHITECOATS . In this chapter we see how we can use two dimensional and three dimensional polymetric views to extract useful information from large data sets.

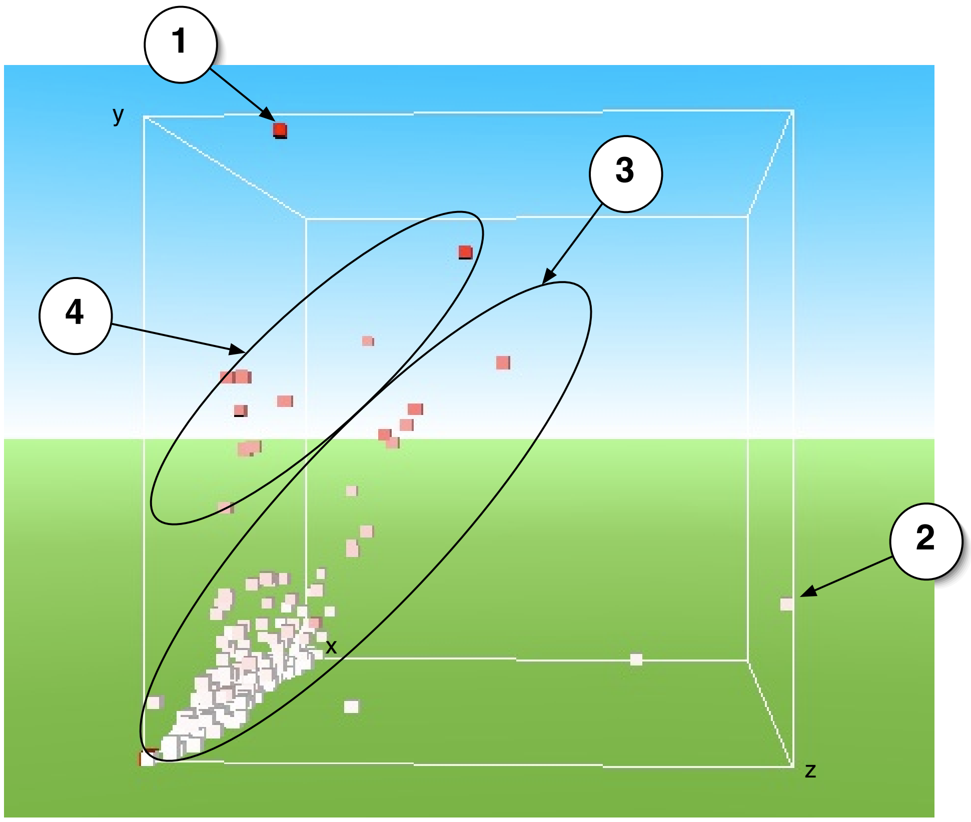
6.1 Outlier Detection



Entities	X	Y	Z	Color
Files	First revision	Added lines	number of revisions	Removed lines

Figure 6.1. Azureus files.

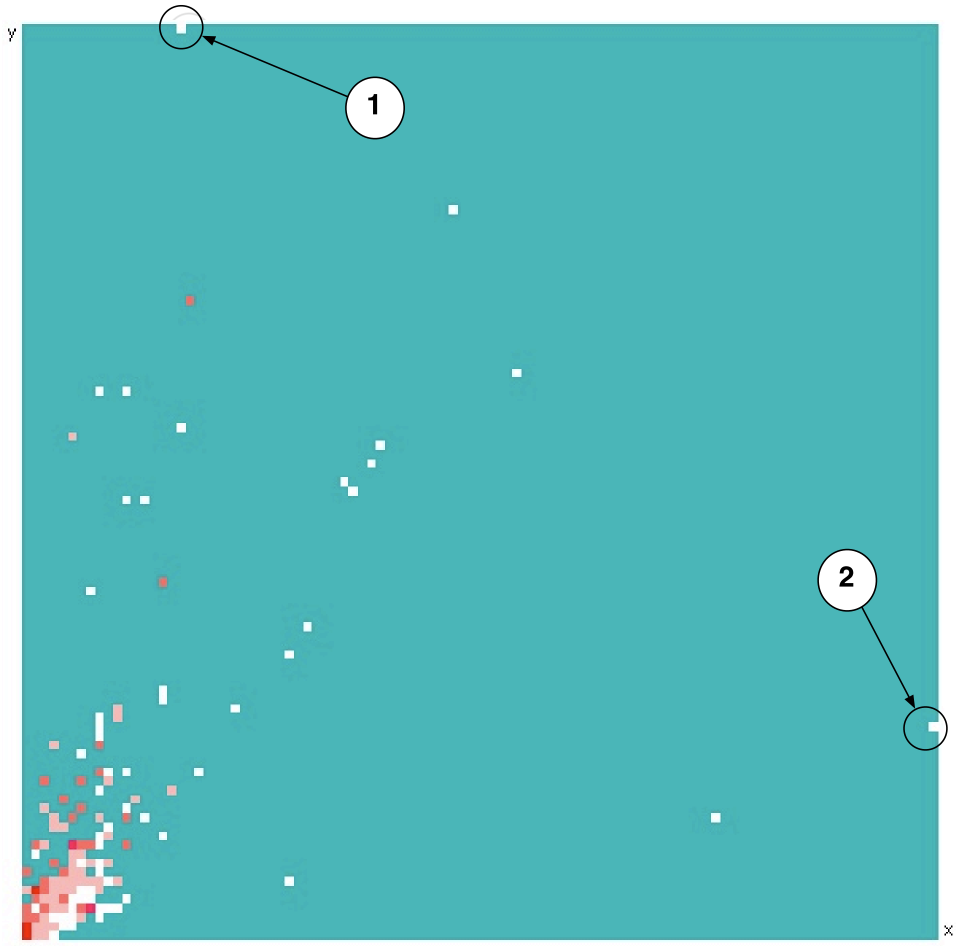
In Figure 6.1 we identify one outlier which is number 2 in the figure. This is a file named "azureus2/org/gudy/azureus2/internat/MessagesBundle.properties". It was created at the beginning of the project and suffered a lot of revisions but with small changes (the number of added lines and removed lines are small). The sets 1 and 3 are interesting, we will discuss them in the case study.



Entities	X	Y	Z	Color
Files	First revision	Added lines	number of revisions	Removed lines

Figure 6.2. Azureus files, rotated.

If we rotate the view we can identify new outliers as we see in Figure 6.2. The files which followed a constant evolution are in the clusters 3 and 4 (files of cluster 4 have suffered stronger changes). The file 2 is the same outlier we already identified in Figure 6.1. The file number 1 named "azureus2/org/gudy/azureus2/internat/MessagesBundle_es_ES.properties", its number of removed lines (color) and its number of added lines (y axis) are high but its number of revisions is small (z axis). What is interesting is that this outlier did not appear in the Figure 6.1.



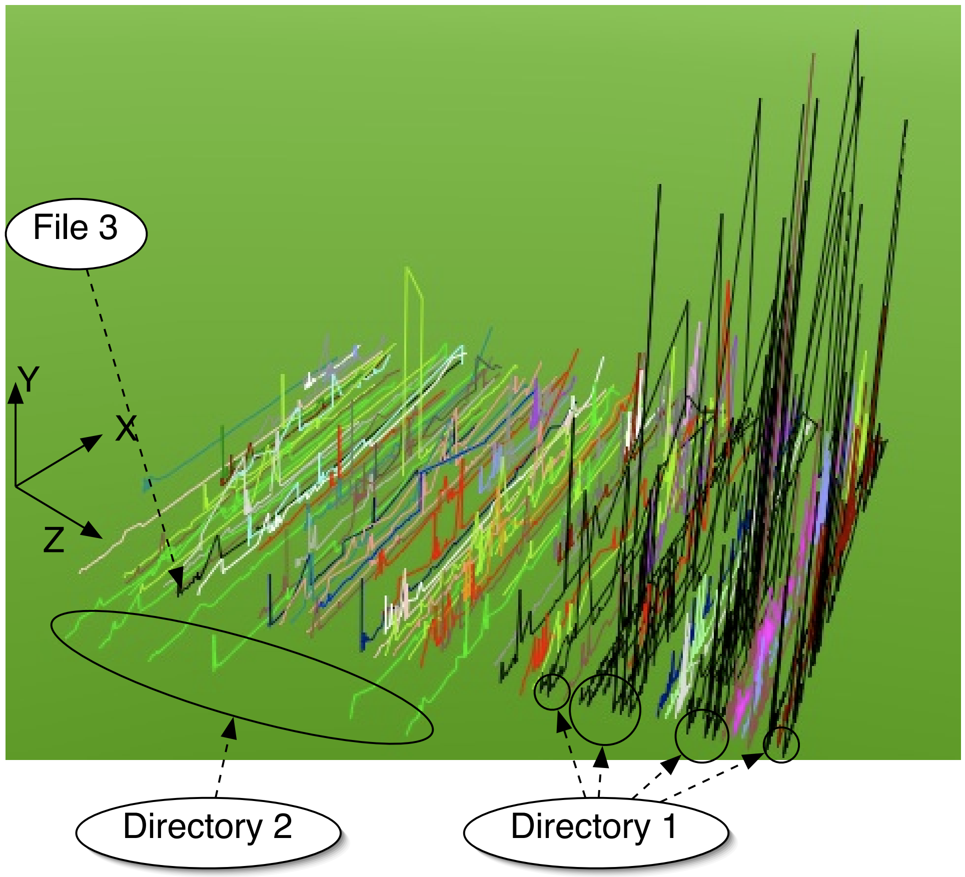
Entities	X	Y	Color
Files	Number of revisions	Added lines	First revision

Figure 6.3. 2d polymetric view of files entities of Azureus.

In Figure 6.3, we see that two dimensions polymetric views is also a way to find outliers, these outliers are the combination of only 3 metrics. But we still find the outliers numbered 1 and 2 in the Figure 6.2.

6.2 Evolution Mining

In Chapter 5 we presented the *subsets threads*. Using these views we find correlated evolutions of files using sequential data. In Figure 6.4 the polylines are files' evolution. Files within the same directory results in polylines with the same color. Using this view we determine directories in which files evolve together (Directory 1 in the figure). The file numbered 3 is also colored in black as all the files within the directory 1, this file is isolated maybe it does not semantically belong to its directory.



Entities	X	Y	Z	Color
Files	Date	Added lines	Sort algorithm	File Directory

Figure 6.4. Evolution Subset threads of Azureus files of more than 50 revisions

The files within directory 2 evolves together but are separated by files from other directories. This could be a tree, directory 2 being the main directory with files and subdirectories. The files within the subdirectories evolves with the files of the main directory.

Chapter 7

A Case Study

For the case study we had to choose a project which met some requirements:

- The project must be available and open-source, so that we have the right to present its information.
- The project must use a versioning system. This versioning system must be CVS.
- The number of revisions must be significant. If not, then the project could be manually analyzed.

Every software which respects these requirements can be analyzed using WhiteCoats. The methodology presents one way of using WhiteCoats, this can be reproduced on other projects but users are still free to use it in another way.

Azureus is an open-source peer-to-peer software available on Sourceforge. It is at the time of the writing of this thesis the most active project on Sourceforge. Sourceforge gives the opportunity to open-source software to use CVS repositories. Azureus has a CVS repository as versioning system which is conform to our requirements.

Azureus comports 3494 files in its repository. These files are organized in 988 directories. A lot of these files and directories are CVS structure components, but still Azureus is big enough to be analyzed.

The files have been written by 22 authors in 16084 revisions. The revisions have been made between the 26 april 2004 and the 22 june 2005. The project lasts for more than a year and evolved enough so we can analyze its evolution.

We first present the methodology we use for this case study. Then an analysis in the large, followed by an analysis in the small.

7.1 Methodology

The methodology presented here is a top-down methodology. We start with large views of the project to go to analysis of small events.

As we can see in Figure 7.1 to analyze a project we must do two analyses. These analyses are complementary and give us different results.

- An analysis in the large.
- An analysis in the small.

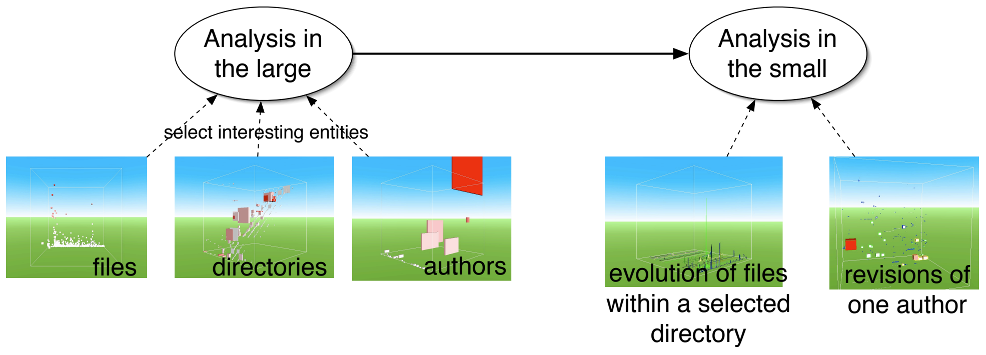


Figure 7.1. Methodology used for this case study.

The analysis in the large is composed of polymetric views of the whole project. We begin with these views to have a better understanding of the project structure. Using WHITECOATS and its CVS module, we can visualize every author of the project, every file and every directory. The polymetric views are designed to handle such sets of entities.

For the analysis in the small, we will use the *dive in* functionality of whiteCoats. We will dive in entities chosen during the analysis in the large and which seems relevant to further analysis. In this analysis we will still use polymetric views on smaller amounts of entities, but also subset threads.

7.2 In the large

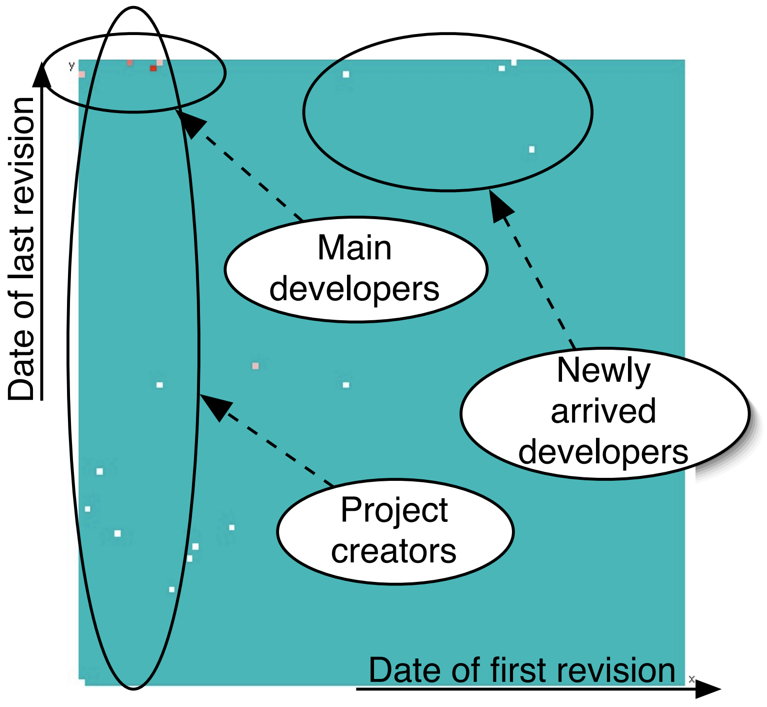
In this section all the views present entities of the full system. Using these views we have an overview of the project and identify special entities for further investigation.

7.2.1 Authors

Azureus has been written by 22 authors. New developers or project managers can use these informations to contact the authors who are able to answer their questions.

Information we want to find:

- Who started the project, and are they still working and available.
- Who are the main developers.
- Who are the new developers.



Entities	X	Y	Color
Authors	First revision	Last revision	Added lines

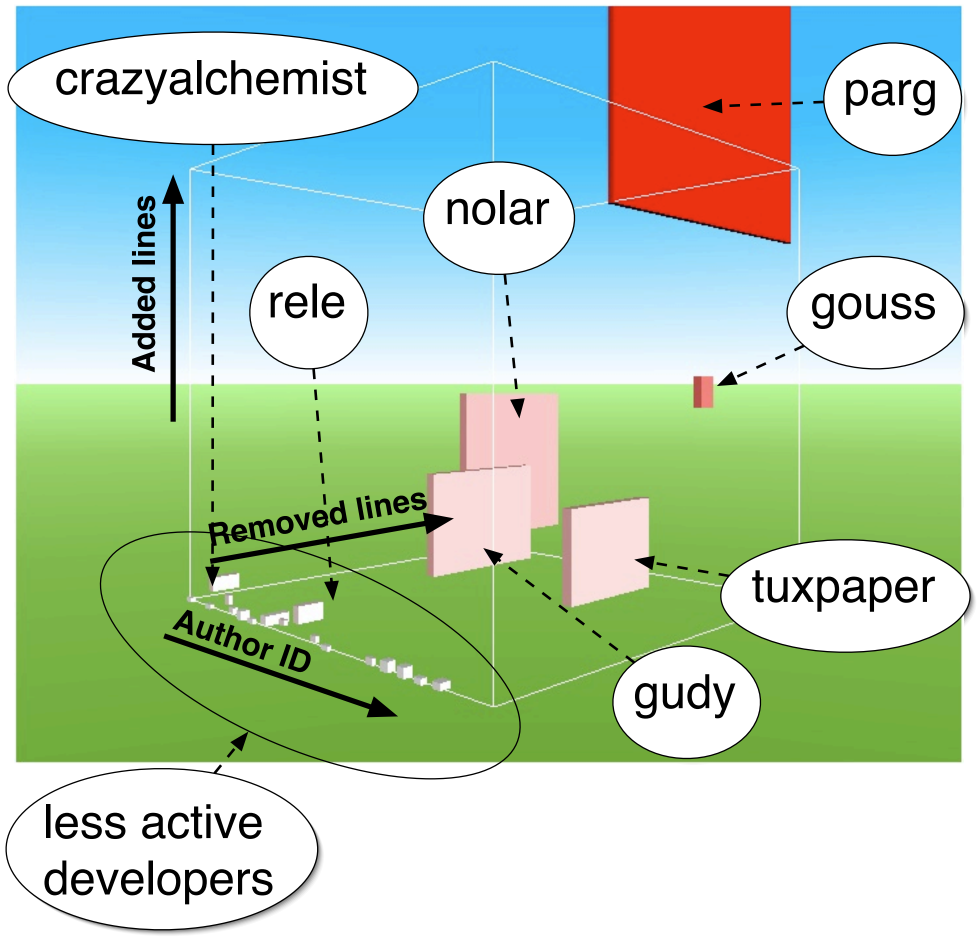
Figure 7.2. Azureus authors time period.

In Figure 7.2, we extract three groups of authors represented by circles.

Main developers in the top-left corner of the view started the project and are still working on it (small date of first revision and maximum date of last revision). It is interesting to notice that these authors are also the most active developers. Every author in this group contributed with at least 20’000 lines of code to 145’000.

Project creators are the main developers plus developers who started the project but stopped to work on it. These authors didn’t contribute a lot, but they certainly did a lot to create the structure of the project.

Newly arrived developers are developers who started to work recently on Azureus. As we see these developers didn’t contribute a lot (between 400 lines to 4’000). This is normal for they worked for a shorter time than the others.



X	Y	Z	Width	Height	Depth	Color
Removed lines	Added lines	Author ID	Number of files	Number of revisions	-	Added lines

Figure 7.3. Azureus authors work activity.

In Figure 7.3, main developers from Figure 7.2 are here the one more colorful, we detail their attributes here.

The authors Parg and Gouss contributed with 200’000 lines on the project.

- Parg worked on 1582 files. He did 8959 revisions in which he added 145’912 lines and removed 65’204 lines. He is the main developer of the project.
- Gouss worked on 55 files, but he is the author who added the most lines after Parg, 66’243. He removed 46’567 lines and did 527 revisions. We can consider he did a very big job on the files he worked on.

The author who follow are bug correctors, they worked on a lot of files but didn’t add a lot of lines, so their revisions must be small changes, corrections or refactoring.

- Nolar, added 26168 lines and removed 20652 lines. He did 2644 revisions on 451 files. Considering the number of files and the number of added lines.
- Tuxpaper worked on 345 files and did 1447 revisions. He added 16905 lines and removed 14845.
- Gudy did 1717 revisions on 430 files. He added 18499 lines and removed 12282.

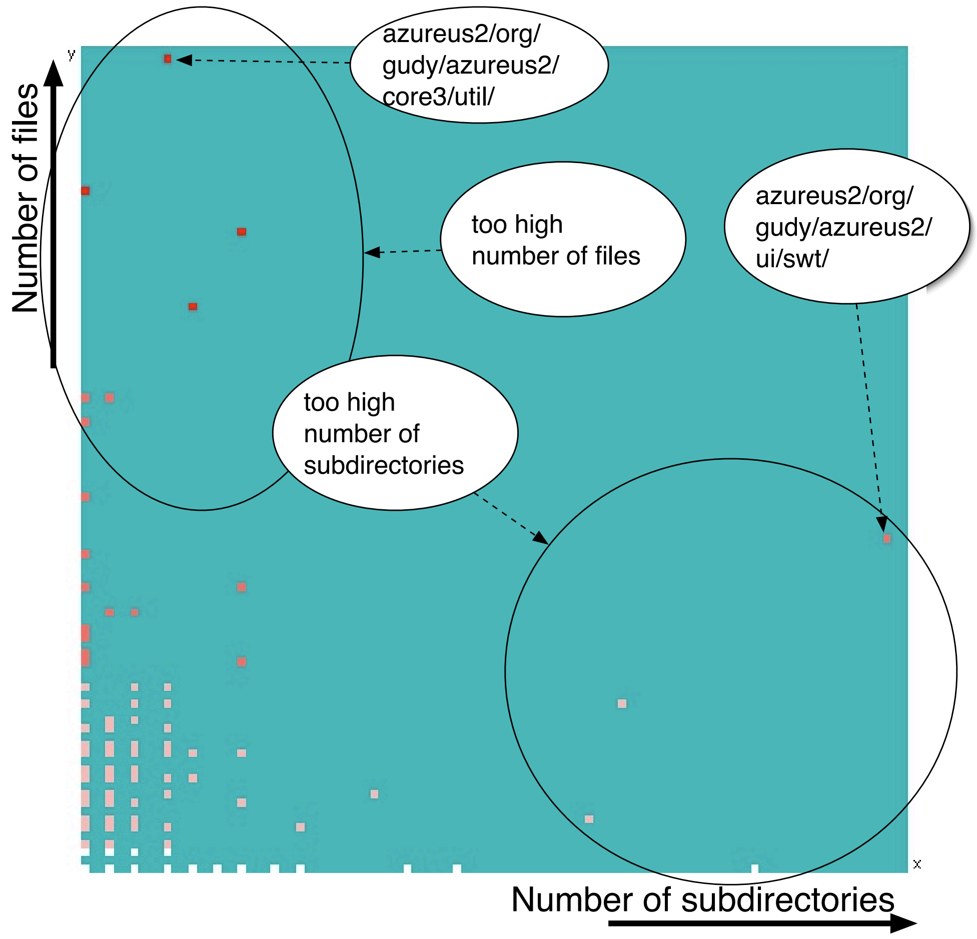
Smaller developers in the bottom-left corner didn’t contribute as much as the others. We extracted two developers who still did a significant work.

- CrazyAlchemist added 3970 lines on 92 files. He removed 2063 lines in 204 revisions.
- Rele worked on 80 files in which he added 3101 lines and removed 1596. He did 235 revisions.

We extracted what we expected from these two views on Azureus authors. We now have groups of developers, we know who started the project, who are the main contributors, the bug correctors and the smaller and new developers. We will do further investigation on some of the authors cited here in the small analysis.

7.2.2 Directories

Azureus is structured in 988 directories. In Figure 7.4, we see that there are no directories with both a large number of files and a large number of subdirectories, the project is well structured.



Entities	X	Y	Color
Directories	Number of subdirectories	Number of files	-

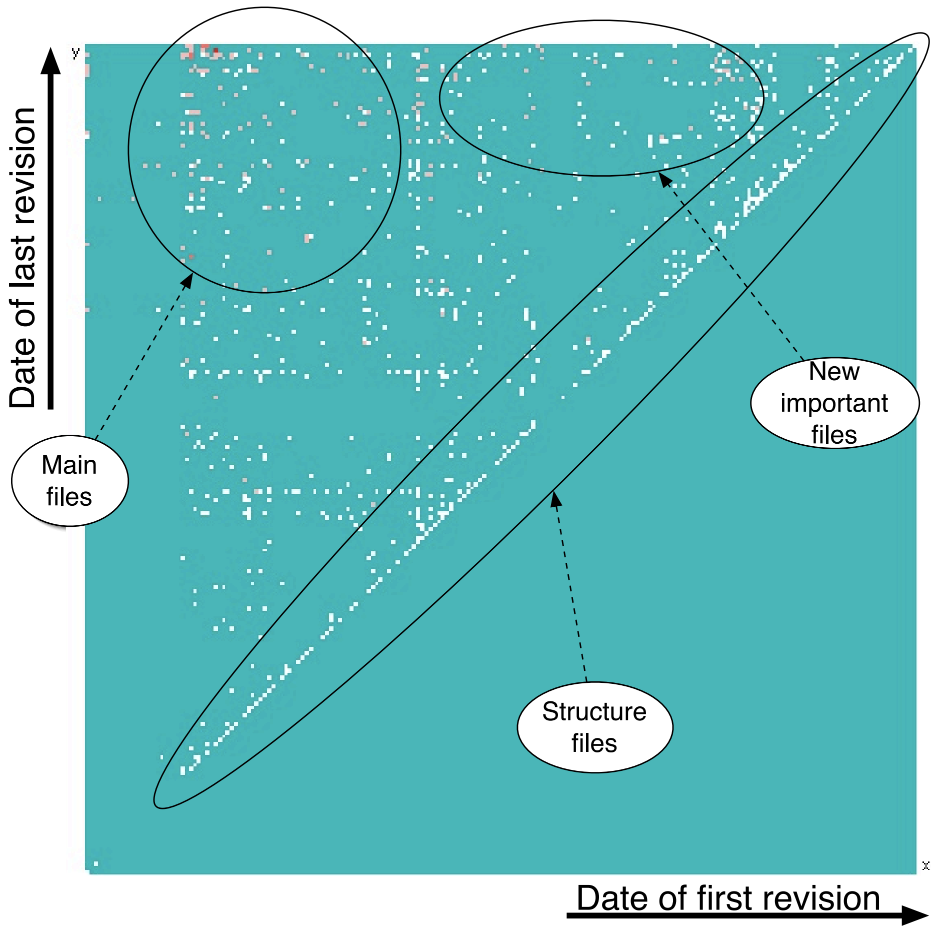
Figure 7.4. Azureus Directory matrix.

Directories with too many subdirectories can be reorganized. It is usually library directories as "azureus2/org/gudy/azureus2/ui/swt/" is the graphic library. But its number of subdirectories is too high (31 subdirectories), and can be reorganized.

Directories with too many files becomes hard to understand for new developers. The directory "azureus2/org/gudy/azureus2/core3/util/" has 65 files which can be splitted into subdirectories. It has only 4 subdirectories.

7.2.3 Files

Azureus is composed of 3494 files. From this analysis, we expect to find too large old files and new files which are becoming too large.



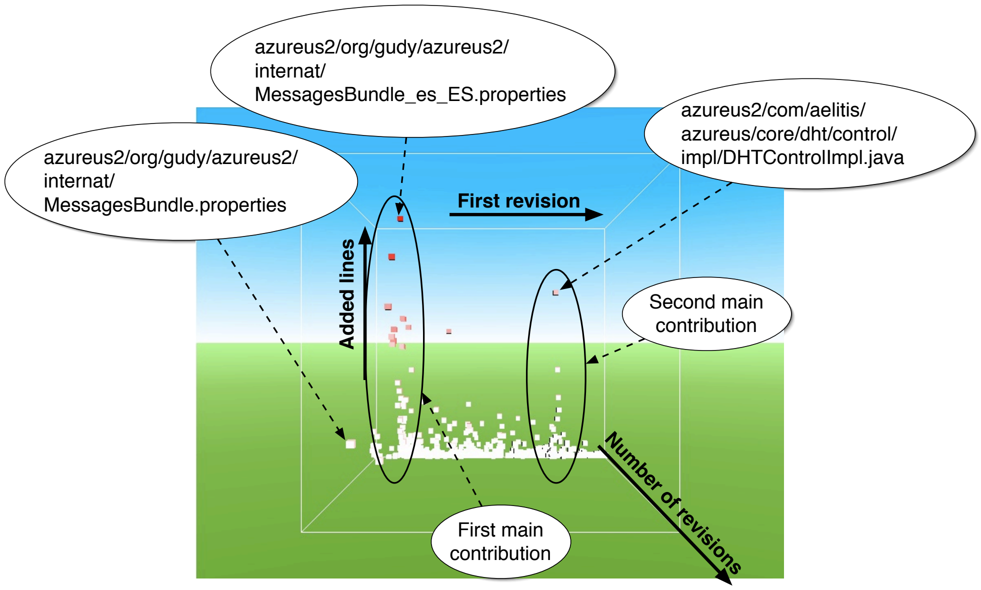
Entities	X	Y	Color
Files	First revision	Last revision	Added lines

Figure 7.5. Azureus files evolution.

In Figure 7.5, we extract three main groups of files:

- Main files are files which were created at the beginning of the project and are still modified today. These files are also the bigger files in the system.
- New important files, have been created recently but are growing very fast.
- Structure files are created and never modified since. These are usually files created by CVS.

In Figure 7.6 the x axis is the date of the first revision, the y axis the number of added lines and the z axis the number of revisions which occurred on the file. The color is the number of removed lines. As we have seen in Chapter 6, we can extract two clusters from this view the same two that we found in Figure 7.5. But in this view we clearly see two phases in the evolution of the system that we call first and second main contributions. We chose two files from each of these phases for the analysis in the small. From the first contribution we choose "azureus2/org/gudy/azureus2/internat/MessagesBundle_es_ES.properties" for further analyze and "azureus2/com/aelitis/azureus/core/dht/control/impl/DHTControlImpl.java", from the second contribution.



Entities	X	Y	Z	Color
Files	First revision	Added lines	Number of revision	Removed lines

Figure 7.6. Azureus files evolution.

Another file is interesting because it was created at the very beginning of the project and is the one with the largest number of revisions : "azureus2/org/gudy/azureus2/internat/MessagesBundle.properties".

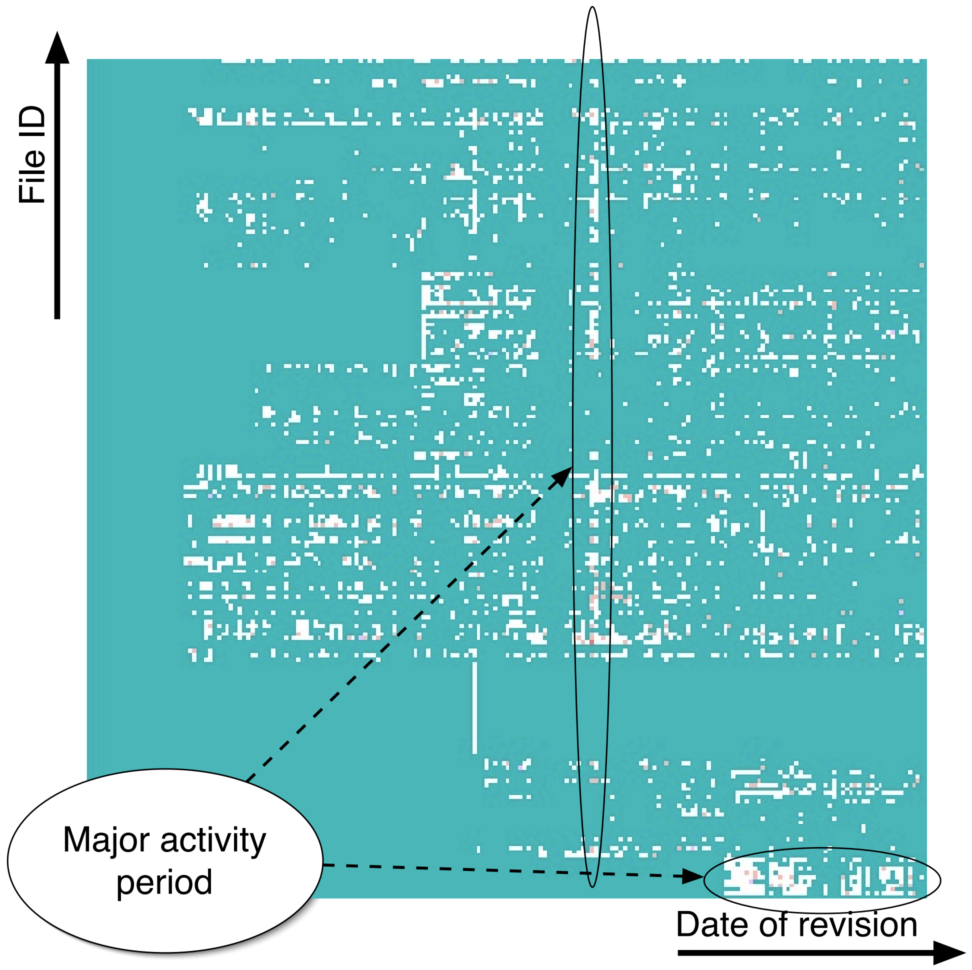
This file may be interesting for further investigation. Its large number of revisions (533) can be the consequence of a lot of bugs or new requirements.

From these two views, we were able to find two phases in the creation of large files. Files from the first phase are actually very large we will analyze one of them in the second part of this analysis. In the second phase some files are becoming too large, we will also analyze one of them.

7.3 In the small

In this section we will analyze some of the entities selected in the previous section. We will focus on the evolution information of these entities. To analyze their evolution we will use the revisions stored by CVS.

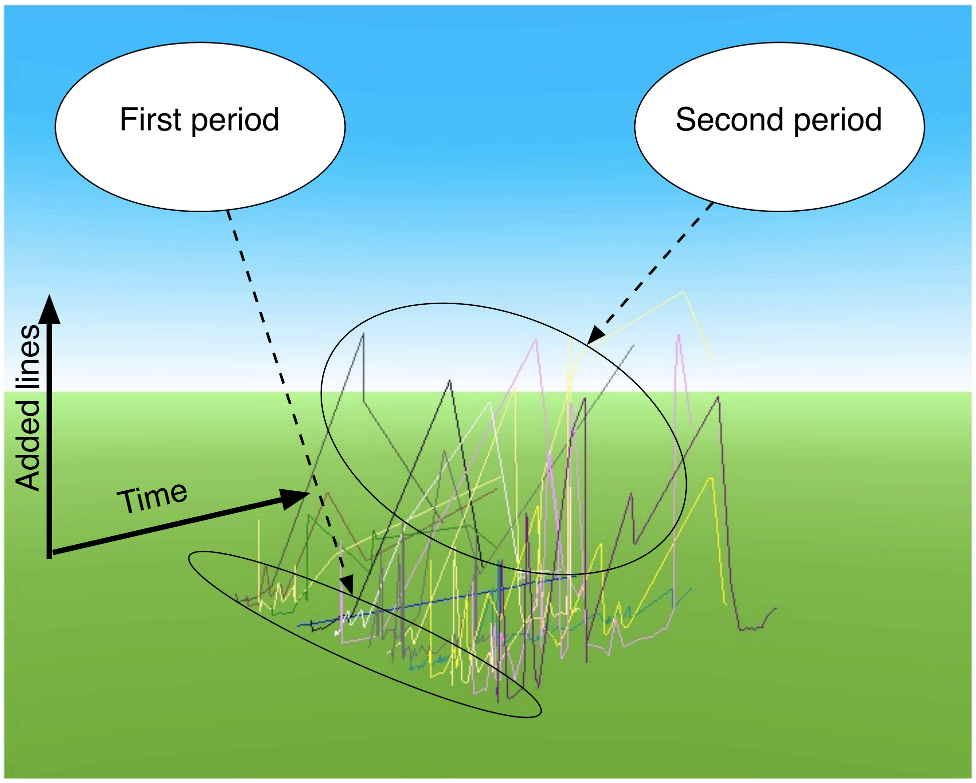
7.3.1 Authors



Entities	X	Y	Color
Parg's Revisions	Date	File ID	Added lines

Figure 7.7. Parg evolution matrix.

The author who worked the most on Azureus is named Parg. In Figure 7.7 we can see the evolution matrix of the files on which Parg worked. As we see Parg did not do a lot of huge revisions as all revisions are white or rose (his bigger revision is of 206 lines). We can see two periods with more work than the rest of the time. The first one is a condensed set of revisions on all files he worked on. The second one in the bottom-right corner is a condensed work on only a module of the system.



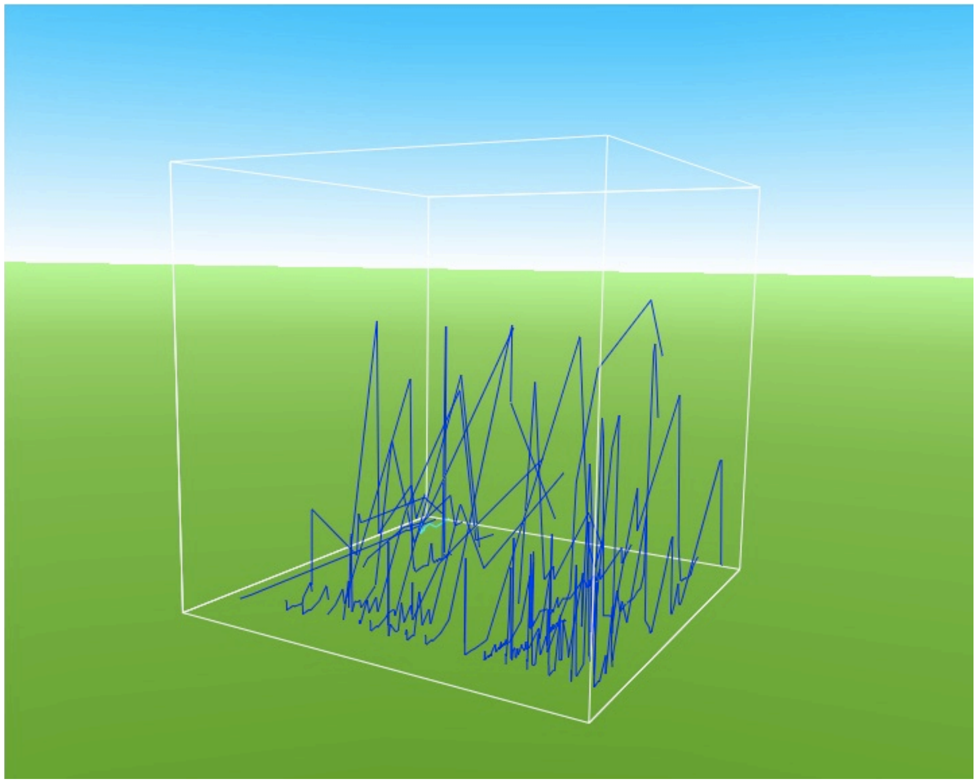
Entities	X	Y	Color
Gouss' Files	Date of revision	Added lines	Random

Figure 7.8. Gouss evolution subsets threads.

In Figure 7.8 We present the subsets threads of the files edited by the author Gouss. The polylines points represent revisions made by Gouss. We can clearly see that when Gouss was starting to work on the project his revisions were smaller (in number of added lines) and more frequent. After that and until now he did bigger revisions with long periods with no revision.

In Figure 7.9 the color of the polylines represents the directory of the associated file. In this view, we see that gouss worked only on one directory.

Parg and Gouss have two different way of working, they are the two main authors of the project.

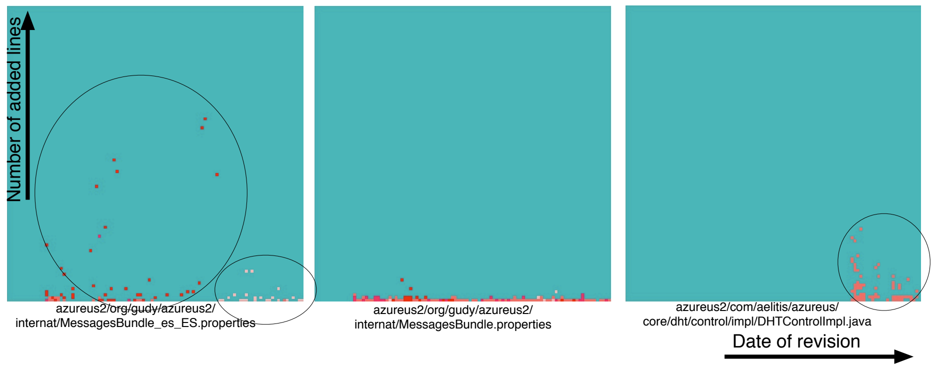


Entities	**X**	**Y**	**Color**
Gouss' Files	Date of revision	Added lines	Directory

Figure 7.9. Gouss evolution subsets threads with directories.

7.3.2 Files

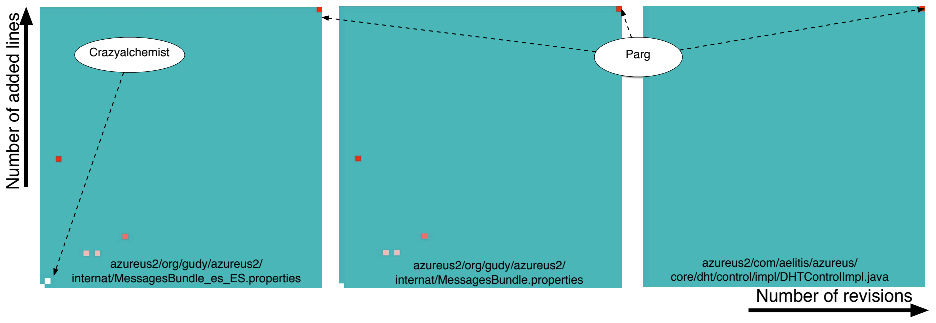
Analyzing files can lead us to know who worked on those files, and how the files have evolve in the project life. From the large analysis we selected three files which needed further investigation.



Entities	X	Y	Color
Files	Date of revision	Added lines	Author ID

Figure 7.10. Revision Matrix of three files.

In Figure 7.10 we show the same three files, but we compare their evolution. We see that the first one had many big revisions made by mainly one author and then all the new revisions are small and by a new author who was not working on this file before. The second file has constantly been revised by many authors, the changes have always been small. The third one is a new file all its revisions have been made by the same author. The size of its revisions is decreasing with time, which means that it is stabilizing.



Entities	X	Y	Color
Authors	Number of revisions	Added lines	Removed lines

Figure 7.11. Authors of three files.

In Figure 7.11 we compare the three files by displaying three views of the authors of each file. The first and the second files have the same authors except that the author Crazyalchemist who is a new developer. The third file has been written only by Parg.

Chapter 8

Conclusion

In this thesis we presented a new visualization tool, WHITECOATS which works on the web. This tool is based on the polymetric views principle that we extend in 3D. We presented how we handle CVS data. We created and used a data-mining technique to improve visualizations. We presented WHITECOATS facilities. We pointed on some visual data mining possibilities. We used a case study to validate our approach.

8.1 Future work

Use flaws. In our model, the relationship between entities is a structural relationship. We would like to add a flaw relation which can relate two or more entities. These flaws can have metrics and should be displayed in the visualizations as edges between entities.

Other modules. We would like to implement other modules to handle other versioning systems information, as SubVersion or StORE.

Improve interaction. Especially with *subsets threads* we have a lack of interaction, there is no diving operation as in polymetric views. In the polymetric views we would like to be able to select multiple entities and then dive into the combination of these entities (dive in revisions of multiple files, files of multiple directories).

Display information. We would like to be able to display information inside a view. For example it will be useful to display the name of selected entities within the visualization. We are currently working on writing text in VRML without surcharging the visualization. Having the same kind of markers we used in the figures of this thesis would be handful for writing analysis.

Apply on other kind of data. In this thesis we presented how we use WHITECOATS to visualize software evolution data. WHITECOATS may be useful to tackle other kinds of research, even more if we implement flaws. We would like to analyze city data, for example traffic-jam problems. Our Evolution Subset Threads view could be applied on any sequential data such as evolution of a sickness of a patient or evolution of the traffic on web-sites in a day.

Bibliography

[1] T. Ball and S. Eick. Software visualization in the large. *IEEE Computer*, pages 33–43, 1996.

[2] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization — Using Vision to Think*. Morgan Kaufmann, 1999.

[3] E. J. Chikofsky and J. H. C. II. Reverse engineering and design recovery: A taxonomy. *IEEE Softw.*, 7(1):13–17, 1990.

[4] P. S. D. Hand, H. Mannila. Principles of data mining. *MIT Press*, 2001.

[5] M. D'Ambros. Software archaeology - reconstructing the evolution of software systems. Master's thesis, Politecnico Di Milano, 2005.

[6] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *ICSM '03: Proceedings of the International Conference on Software Maintenance*, page 23, Washington, DC, USA, 2003. IEEE Computer Society.

[7] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, pages 213–228, Fall 1992.

[8] B. Gulla. Improved maintenance support by multi-version visualizations. In *Proceedings of the 8th International Conference on Software Maintenance (ICSM 1992)*, pages 376–383. IEEE Computer Society Press, Nov. 1992.

[9] M. Jazayeri. On architectural stability and evolution. In *Reliable Software Technlogies-Ada-Europe 2002*, pages 13–23. Springer Verlag, 2002.

[10] M. Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *Proceedings of IWPSE 2001 (International Workshop on Principles of Software Evolution)*, page to be published, 2001.

[11] M. Lanza and S. Ducasse. Understanding software evolution using a combination of software visualization and software metrics. In *Proceedings of LMO 2002 (Langages et Modèles à Objets*, pages 135–149, 2002.

[12] M. Lanza and S. Ducasse. Polymetric views — a lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29(9):782–795, Sept. 2003.

[13] C. Mesnage and M. Lanza. White coats: Web-visualization of evolving software in 3d. In *VISSOFT : IEEE International Workshop on Visualizing Software for Understanding and Analysis*. To be published, 2005.

[14] R. Robbes and M. Lanza. Versioning systems for evolution research. In *IWPSE '05 : International Workshop on Principles of Software Evolution*, pages 155–164, 2005.

[15] F. V. Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04)*, pages 328–337, Washington, DC, USA, 2004. IEEE Computer Society.

[16] J. T. Stasko, J. Domingue, M. H. Brown, and B. A. Price, editors. *Software Visualization — Programming as a Multimedia Experience*. The MIT Press, 1998.

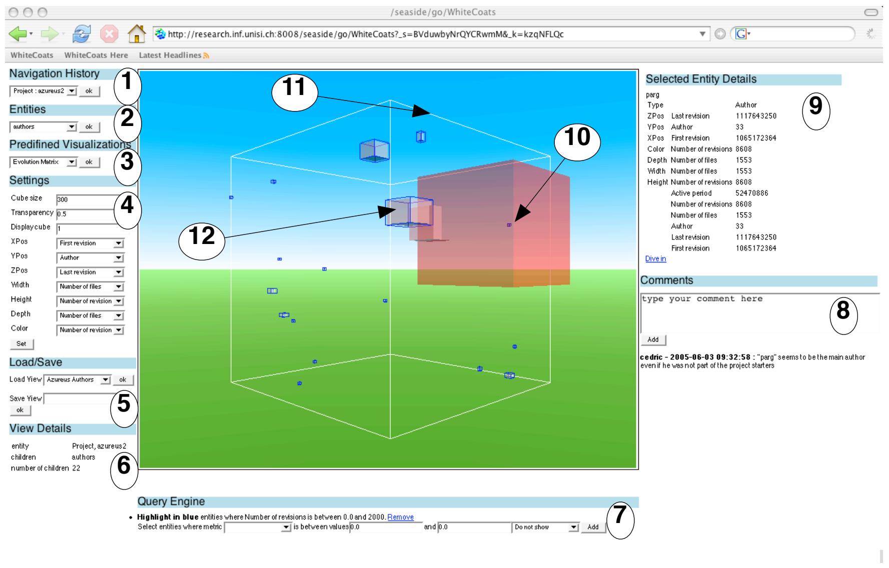
[17] C. M. B. Taylor and M. Munro. Revision towers. In *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, pages 43–50. IEEE Computer Society, 2002.

[18] J. Wu, R. C. Holt, and A. E. Hassan. Exploring software evolution using spectrographs. In *11th Working Conference on Reverse Engineering (WCRE'04)*, pages 80–89, November 2004.

List of Figures

Appendix A

White Coats User Interface


/seaside/go/WhiteCoats
http://research.inf.unisi.ch:8008/seaside/go/WhiteCoats?_s=BVduwbyNrQYCRwmM&_k=kzqNFLQc
WhiteCoats
WhiteCoats Here
Latest Headlines
1
2
3
4
5
6
7
8
9
10
11
12

Navigation History

Project : azureus2 ok

Entities

authors ok

Predifined Visualizations

Evolution Matrix ok

Settings

Cube size	300
Transparency	0.5
Display cube	1
XPos	First revision
YPos	Author
ZPos	Last revision
Width	Number of files
Height	Number of revision
Depth	Number of files
Color	Number of revision

Set

Load/Save

Load View Azureus Authors ok

Save View ok

View Details

entity	Project, azureus2
children	authors
number of children	22

Selected Entity Details

parg

Type		Author
ZPos	Last revision	1117643250
YPos	Author	33
XPos	First revision	1065172364
Color	Number of revisions	8608
Depth	Number of files	1553
Width	Number of files	1553
Height	Number of revisions	8608
	Active period	52470886
	Number of revisions	8608
	Number of files	1553
	Author	33
	Last revision	1117643250
	First revision	1065172364

Dive in

Comments

type your comment here

Add

cedric - 2005-06-03 09:32:58 : "parg" seems to be the main author even if he was not part of the project starters

Query Engine

- **Highlight in blue** entities where Number of revisions is between 0.0 and 2000. Remove

Select entities where metric is between values 0.0 and 0.0 Do not show Add

Figure A.1. The user interface of WhiteCoats.

The user interface of *White Coats* (see Figure A.1) is composed of one central 3D viewport (navigable and interactive) and many panels that provide complementary information:

1 Navigation History: This panel gives the user the ability to navigate to previously visualized entities. Diving into an entity adds that entity to the history list.

2 Entities: When an entity is visualized, the user selects the kind of entities related to it that he wants to display. For example if a directory is the main entity, the contained subdirectories or files can be displayed.

3 Predefined Visualizations: A set of applicable, predefined visualizations (*e.g.,* ”Evolution Matrix”). When one is selected, the other panels are updated accordingly.

4 Settings: A visualization can be configured by setting the associations between visual attributes (*e.g.,* width, height, position, *etc.*) and metrics (*e.g.,* ”Number of files”) or by changing the size of the reference cube and the transparency of the entities (useful when they overlap).

5 Load/Save: Saving a view stores its configuration, *e.g.,* entities displayed, metrics settings, *etc.*. The saved view is shared with other users exploring the same project. Loading a view updates all panels.

6 View Details: A summary of the displayed data, such as the name of the visualized entity, the type of entities displayed, and the number of entities in the visualization.

7 Query Engine: One can add queries to filter the displayed entities. By setting a lower and upper threshold on a metric entities are selected on which an action is performed (*e.g.,* ”Do not show” or ”Highlight”).

8 Comments Panel: Users can read and write comments associated to an entity. The comments are displayed with a date and the comment’s author user name.

9 Selected Entity Details: When a user clicks on an entity, it displays related information, such as its name, the metrics associations, *etc.*. A ”Dive in” link makes the selected entity become the main entity, *i.e.,* the internals of that entity are displayed.

10 A visualized entity: An entity is displayed as a box in the visualization, its position, size and colors depending on the associated metrics.

11 The reference cube: A transparent wireframe containing all displayed entities. The reference cube eases 3D navigation. By setting the size of the cube, it scales the contained boxes.

12 A highlighted entity: When an entity is highlighted because if matches a query it displays a wireframe box around it. The wireframe color can be set.

A.1 3d facilities

A.1.1 Reference Cube.

The displayed entities are displayed within a surrounding wireframe cube. This *reference* cube gives an idea of the orientation to the user. We can also set the size the cube, giving the user the ability to spread the entities if they overlap. As we see in Figure A.2, if the *reference cube* is not displayed (right screenshot), we have no clue on the position of the entities.

A.1.2 Horizon.

As a supplemental navigation aid we also provide a ”sky” and a ”ground” background to let the viewer know into which direction he is looking.

A.2 Query engine

The query engine is one of the user interface component which allow the user to execute actions on the entities he selects.

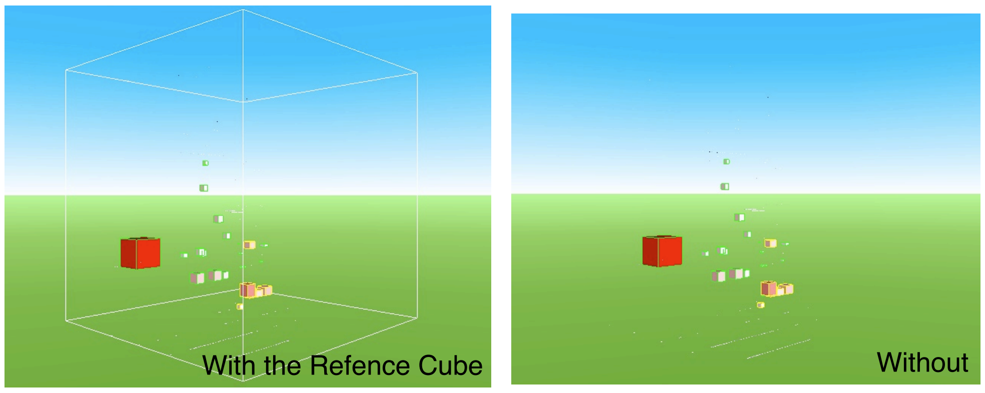


Figure A.2. The importance of the reference cube.

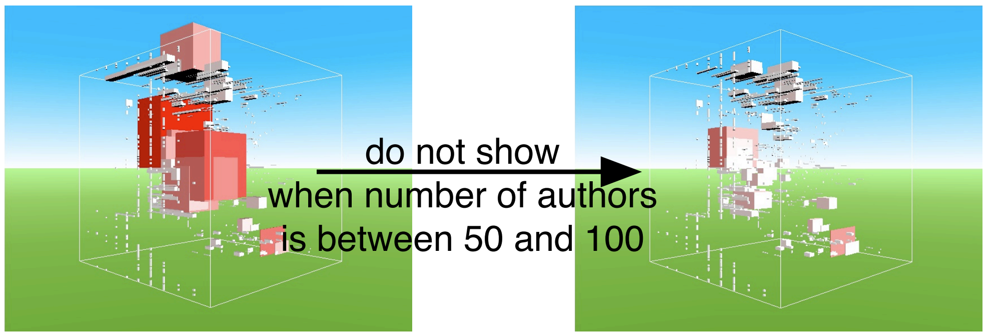


Figure A.3. Restricting the view.

A.2.1 Restriction on the metrics

After selecting a metric on which apply the restriction, the user will have to set a minimum and a maximum. The resulting query is equivalent to :

Given a metric, a minimum and a maximum : $\forall$ entities to visualize , select entity where min <value of metric for entity $< max$.

A.2.2 Associate actions to constraints

Once the entities are selected by a query, an action will be executed on them. For now, two type of actions are available, the *do not show* action and the *highlight* actions.

Do not show If an entity is selected by a query which action is *do not show* then the selected entities are removed from the entities to display. Then we have :

$$entitiesToDisplay = entitiesToDisplay - entitiesSelected$$

Highlight entities If the action is an *highlight* action which can be of different colors (blue,

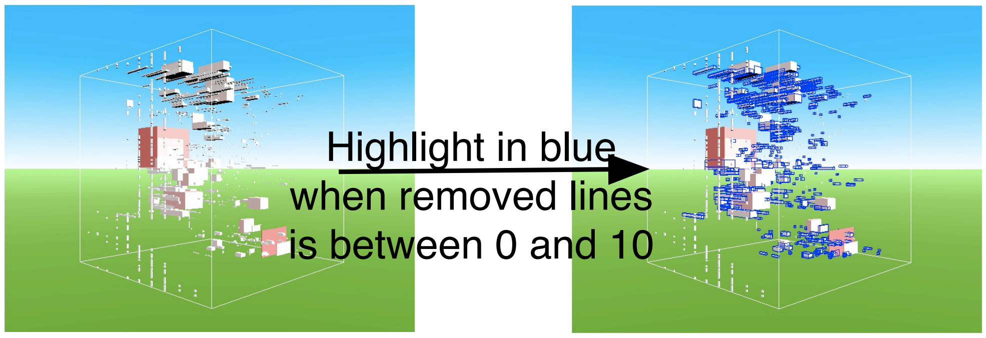


Figure A.4. Highlighting entities.

green or yellow), the selected entities will results into figures with an highlight cube around it (or highlight square in 2d).

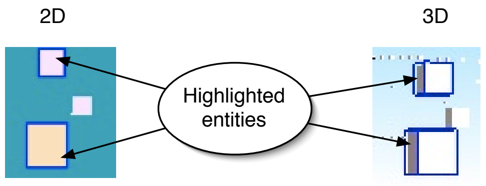


Figure A.5. 2D and 3D example of highlighted entities.

A.3 Interaction

White Coats uses VRML for the visualizations, whereas navigation and interactivity are given by the used VRML browser plugin [1] we use. This plugin allows to navigate in the view by rotating around the objects or flying into it. We defined a set of viewpoints (front, back, bottom, right side, left side) which can be chosen during the navigation.

The visualized entities can be selected using the mouse pointer. When clicked, the user can dive into the entity, inspect it, *etc.*. In case of a diving, a new visualization is created, *i.e.,* a new VRML document is created on the fly and displayed.

In the Figure A.6 we see how the user can *dive* into an entity (here a directory). The configuration stay the same after the diving so the newly displayed entities are subdirectories of the selected entity.

[1] In our case, we use the Cortona plugin. Accessible at: http://www.parallelgraphics.com/products/cortona/

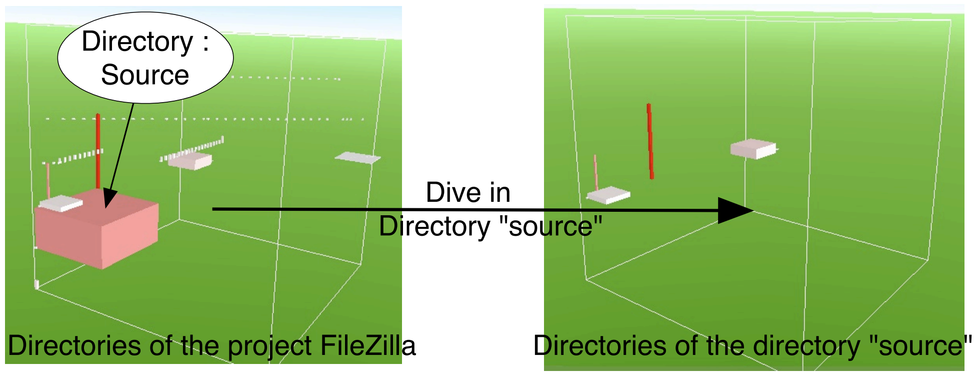

Directory : Source
Dive in Directory "source"
Directories of the project FileZilla
Directories of the directory "source"

Figure A.6. An example of the *Dive in* operation.

A.4 Configuration

The *View settings* panel is where all configuration of the view are done. Using different selection boxes or text entries, the user is able to select what kind of entities he wants to visualize and how to render it.

A.4.1 Configuring the view

The first choice is to select a 2D or 3D view, this is done with a radio button. Then the user can select the type of entity he wants to visualize within the entities which are to be displayed.

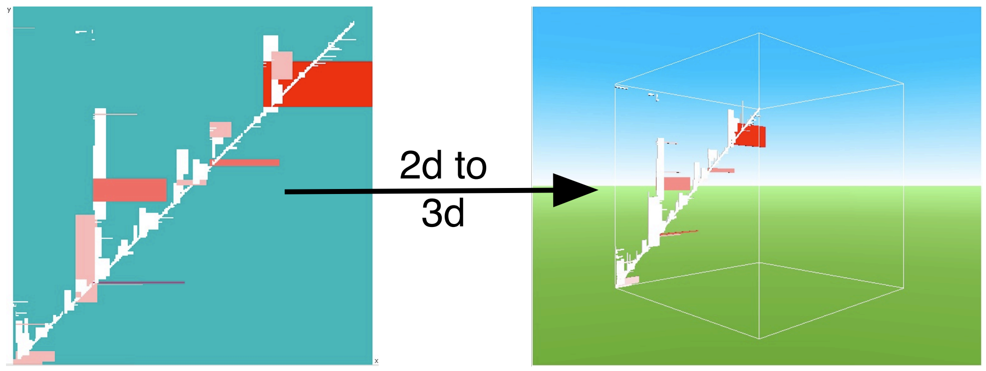


Figure A.7. Passing from a 2D view to a 3D view without changing the configuration.

At any moment the user can pass from a 2D view to a 3D view with the same configuration and vice-versa. In Figure A.7 the configuration sets the x position of the displayed directories of Azureus to the ”parent directory id”, the y axis to the ”directory id”. The width and the height are set to ”Number of subdirectories” and ”Number of files in directory”, the color is the ”number of subdirectories”. The view size is set to 300.

In Figure A.8, setting the view size will increase the spare space between entities, without changing their size. This is very useful when entities are positioned close to each other.

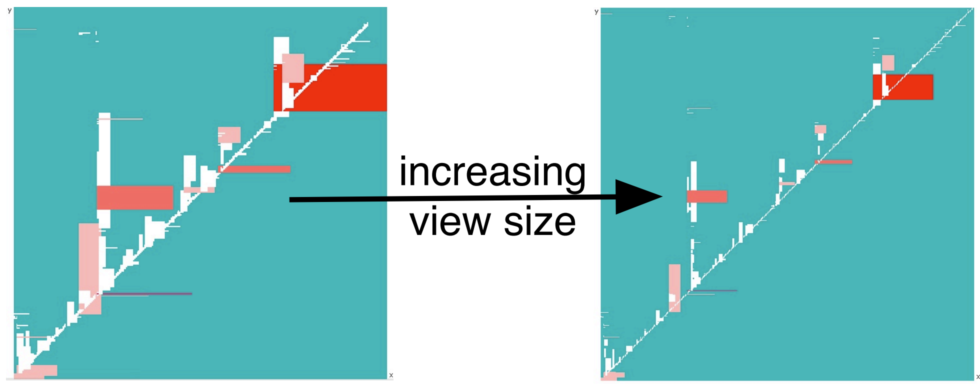


Figure A.8. Setting the view size from 300 to 600.

A set of boolean choices are available. If the visualization is in 3D, we can display the *reference cube* or not. The entities can be displayed, or the polylines (*subsets threads*). And if the lines are displayed, we can chose to sort them by similitude (using the technology we have seen in Chapter 5) or not.

A.4.2 Configuring the metrics

To each dimension, the user have to associate a metric which will be used on each entity to create the resulting figure. An alternative choice is to leave that field blank. Then it means no metric is associated to the dimension and the minimum value will be used (1 for width height and depth, 0 for the others).

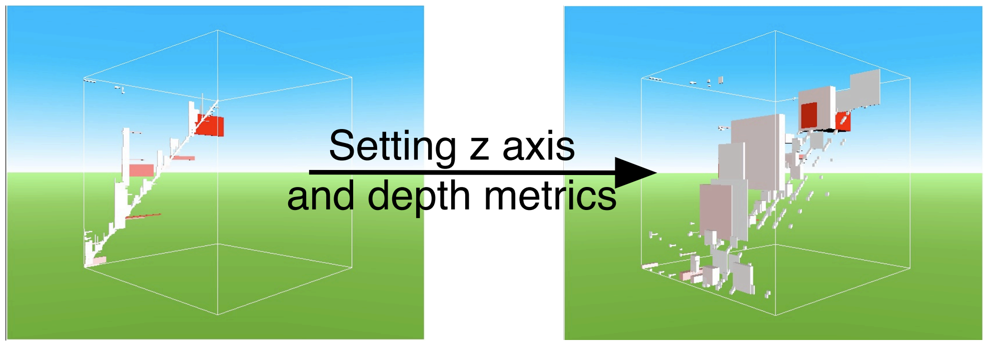


Figure A.9. By configuring the metrics, we change the view.

A.5 Collaborative facilities

Analyzing large systems require many people, software engineers and software reengineers. For their analysis to be successful, they must share their results by writing reports, but they can also share partial results during the reverse engineering process.

WHITECOATS facilitate the collaborative analysis of large systems by giving the ability to the users to share the views they create, and to comment the entities they visualize.

A.5.1 Saved views

A saved view stores all the configuration of the view, the entities visualized and the queries applied on them. Once saved, any other user can automatically load this view and visualize it.

A.5.2 Comments

When visualizing any entity, the user can add a comment to this entity which will be shared with other users when they will visualize this entity.

INTERACTIVE AND COOPERATIVE VISUAL DATA MINING OF EVOLVING SOFTWARE

CÉDRIC MESNAGE

The maintenance, reegineering, and evolution of software systems has become a vital matter in today's software industry. With time systems tend to decay in quality. Such legacy software systems need to be reverse engineered in order to keep their value. One important source of information that is seldom used is the history of the system, i.e., the life-cycle it went through until its current state.

In this master thesis we propose a **visual data mining approach** to **reverse engineer** systems using the history data provided by **versioning systems**. We extract **software entities** from CVS (Concurrent Versioning System) and maintain these in a **data warehouse**.

Our visualizations are based on the **polymetric views** principle that we extend in **3D** and enrich with **data mining** techniques.

We implemented these visualizations in WHITECOATS , an interactive and cooperative tool working as a web-service, which we use to validate our approach in a case-study.