

Diploma Thesis

April 25, 2007

# Visualizing Dynamic Social Network Structures

An Evaluation of an Open Source Software  
Development Community

**Barbara Schwarz**

of Winterthur, Switzerland (02-714-541)

**supervised by**

Prof. Harald Gall

Dr. Martin Pinzger



University of Zurich  
Department of Informatics





Diploma Thesis

---

# Visualizing Dynamic Social Network Structures

An Evaluation of an Open Source Software  
Development Community

Barbara Schwarz



University of Zurich  
Department of Informatics



**Diploma Thesis**

**Author:** Barbara Schwarz, [barbara\\_schwarz@access.unizh.ch](mailto:barbara_schwarz@access.unizh.ch)

**URL:** [ifi.unizh.ch/seal](http://ifi.unizh.ch/seal)

**Project period:** 25 October 2006 - 25 April 2007

Software Evolution & Architecture Lab

Department of Informatics, University of Zurich

---

# Acknowledgements

I would like to thank Prof. H. Gall for giving me the opportunity of writing my diploma thesis at the S.E.A.L. group. Special thanks go to my supervisor Dr. Martin Pinzger who supported me during the last six months.

I would like to thank all my friends, my team-mates and training companions for making my mind off things and for the interesting discussion. Special thanks go to my proofreaders.

A special thank goes to Dr. Peter A. Gloor, who allowed me to use his iQuest tool.

Last but not least, I would like to thank my parents, especially my father who always supported me in achieving my objectives.



---

# Abstract

Since large and complex software projects consist of people playing different roles, the communication and collaboration among the team members are key success factors. This interaction is reflected in the organization and has an influence on the project and its outcome. High-quality Open Source Software (OSS) relies on having a large and sustainable community. This implies that community members must understand the projects environment.

We aim at providing a visualization prototype tool that allows the project members to interactively explore the project dynamics. This visualization extends social network graphs by work packages representing software contribution. The visualisation approach is evaluated on the Eclipse OSS community. It is shown that different scenarios and roles can be discovered, providing deeper insight into the project.





---

# Zusammenfassung

Die Mitarbeiter von grossen und komplexen Software Projekten nehmen unterschiedliche Rollen ein. Deren Kommunikation und Zusammenarbeit spielen eine entscheidende Rolle fuer den Erfolg von Open Source Software (OSS) Projekten. Diese Interaktionen spiegeln sich nicht nur in der Organisation wieder, sondern beeinflussen auch die Strukturen der Software. Dabei bedingt qualitativ gute Software eine stabile und nachhaltige Community. Dies erfordert, dass die Mitglieder die Kultur und das Umfeld des Projektes genau kennen.

Wir fuehren ein Visualisierungstool ein, welches den Projektmitgliedern erlaubt, die Entwicklung des Projektes interaktiv zu erforschen. Dazu werden Soziale Netzwerke um Aenderungsbeitrge erweitert. Der Ansatz wird am Beispiel der Eclipse OSS Community evaluiert, wobei verschiedene Szenarien und Rollen beobachtet werden knnen. Dies verbessert das Verstaendnis fr das Projekt aus unterschiedlichen Perspektiven.



---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Scope . . . . .	2
1.3	Objectives . . . . .	2
1.4	Outline . . . . .	2
<b>2</b>	<b>Related Work</b>	<b>3</b>
2.1	Social Networks Analysis . . . . .	3
2.1.1	Communication Analysis . . . . .	3
2.1.2	Topological Analysis . . . . .	4
2.2	Socio-Technical Networks Analysis . . . . .	4
2.3	Software Development Teams . . . . .	5
2.3.1	Software Evolution . . . . .	5
2.3.2	Open Source Software . . . . .	5
2.4	Visualization . . . . .	5
<b>3</b>	<b>Interaction in Software Projects</b>	<b>7</b>
3.1	Perspectives . . . . .	7
3.1.1	Perspective: Project Manager . . . . .	8
3.1.2	Perspective: Newcomer . . . . .	8
3.2	Social Network Analysis Cockpit . . . . .	8
3.3	Understanding Project Collaboration . . . . .	10
3.3.1	Software Development . . . . .	10
3.3.2	Communication Channels . . . . .	10
3.4	Interaction Target Model . . . . .	11
3.4.1	Software Evolution Model . . . . .	12
3.4.2	Communication Model . . . . .	15
3.5	Summary . . . . .	18
<b>4</b>	<b>Data Preparation</b>	<b>19</b>
4.1	Data Processing . . . . .	19
4.2	Data Extraction Tools . . . . .	20
4.2.1	Evolizer CVS Importer . . . . .	20
4.2.2	iQuest Mailing List Parser . . . . .	23
4.2.3	Bugzilla Parser . . . . .	25
4.3	Data Integration . . . . .	27
4.3.1	Person Allocation . . . . .	28

4.3.2	Delimitation & Timeframe Alignment . . . . .	31
4.4	Summary . . . . .	31
<b>5</b>	<b>Visualization</b>	<b>33</b>
5.1	Visualization Styles . . . . .	33
5.1.1	Visualizing Time-Dependency . . . . .	33
5.1.2	Visualizing Collaboration Patterns . . . . .	34
5.1.3	Visualizing Social Networks . . . . .	35
5.1.4	Motivation . . . . .	36
5.2	Dimensions & Abstraction . . . . .	36
5.2.1	Time . . . . .	36
5.2.2	Configuration . . . . .	36
5.3	Concepts . . . . .	37
5.3.1	Social Network Graph . . . . .	37
5.3.2	Collaboration Network Graph . . . . .	38
5.3.3	Dynamics - Time Window Selection . . . . .	38
5.4	Social Network Analysis Cockpit . . . . .	38
5.4.1	Graphs . . . . .	38
5.4.2	The User Interface . . . . .	43
5.5	Summary . . . . .	44
<b>6</b>	<b>Case Study &amp; Evaluation</b>	<b>45</b>
6.1	Project Selection . . . . .	45
6.1.1	Requirements . . . . .	45
6.1.2	Project Assessment . . . . .	46
6.1.3	The Eclipse Project . . . . .	46
6.2	Project Setup . . . . .	48
6.2.1	Community . . . . .	48
6.3	The Eclipse Platform Core Development Team . . . . .	49
6.3.1	Patterns . . . . .	50
6.3.2	Perspective: Project Manager . . . . .	56
6.3.3	Perspective: Newcomer . . . . .	59
6.4	Summary . . . . .	60
<b>7</b>	<b>Summary &amp; Conclusion</b>	<b>63</b>
7.1	Summary & Conclusion . . . . .	63
7.2	Contribution . . . . .	63
7.3	Limitations . . . . .	64
7.4	Future Work . . . . .	64
<b>A</b>	<b>Source Code Ownership</b>	<b>65</b>
A.1	Revision Characteristics . . . . .	65
A.1.1	Overall Contribution of the Revision's Author . . . . .	65
A.1.2	Person Owning the Relating File . . . . .	66
A.1.3	Overall Contribution of the Owning Person . . . . .	66
<b>B</b>	<b>Data Preparation</b>	<b>67</b>
B.1	Problem Reports . . . . .	67
B.1.1	Bugzilla Importer . . . . .	67
B.2	The Integrated Data Model . . . . .	67
B.3	Data Source Consolidation . . . . .	70

---

<b>C</b>	<b>Social Network Analysis Cockpit Implementation</b>	<b>75</b>
C.1	Implementation . . . . .	75
C.1.1	Problem Report Classification . . . . .	75
C.1.2	org.seal.snanalyzer.evaluation . . . . .	76
C.1.3	Test Cases . . . . .	76
C.1.4	Program Start . . . . .	77
<b>D</b>	<b>The Eclipse Project</b>	<b>79</b>
<b>E</b>	<b>Content of CD-ROM</b>	<b>83</b>

## List of Figures

3.1	Interaction Network . . . . .	9
3.2	High Level Integration Concept . . . . .	11
3.3	High Level Interaction Model . . . . .	12
3.4	Correlation between the File Size and the Frequency of Modification . . . . .	15
3.5	A Mail Thread of an Online Archive . . . . .	16
3.6	The Adjusted Mail Thread with the Corresponding Communication Paths . . . . .	16
3.7	Communication within Problem Reports of Bugzilla . . . . .	17
4.1	Data Source Integration Process . . . . .	20
4.2	Evolizer CVS Importer: Versioning Model . . . . .	21
4.3	iQuest Mail Communication Schema . . . . .	24
4.4	Communication Model . . . . .	26
5.1	Distribution of Collaboration Actions . . . . .	34
5.2	Source Code File Modification over Time . . . . .	34
5.3	iQuest Social Network Graph . . . . .	35
5.4	Social Network Graph . . . . .	39
5.5	SNA Cockpit Analysis Panel . . . . .	42
5.6	SNA Cockpit Control . . . . .	43
5.7	The Modification Reports' Detail Display . . . . .	44
6.1	Eclipse Platform Core Community . . . . .	49
6.2	Connectors of the Eclipse Platform Core Team as of 2003-11-03 . . . . .	50
6.3	Communication from May 2004 to February 2006 . . . . .	51
6.4	Conway's Law: Governance Structure Impacts' View as of 2006-04-13 . . . . .	52
6.5	Snippets of Figure 6.4 . . . . .	53
6.6	More Swiss Contribution Snippets . . . . .	53
6.7	Clean Up Revisions at 2005-02-21 . . . . .	54
6.8	Clean Up Revisions at 2006-05-08 . . . . .	55
6.9	A Project Manager's View as of 2006-04-10 . . . . .	56
6.10	Snippets of Figure 6.9 . . . . .	57
6.11	Rafael Leaving the Core Development Team . . . . .	58
6.12	Collaboration Network as of October 2005 . . . . .	59
6.13	Socialization of Kevin . . . . .	60
B.1	Detailed Data Model . . . . .	69
C.1	Starting the Eclipse Plug-in View . . . . .	77
C.2	Social Network Analysis Cockpit . . . . .	78
D.1	Contribution to Source Code Development over Time . . . . .	81

## List of Tables

3.1	Comparison with Girba's Approach . . . . .	14
4.1	Project Data Consolidation . . . . .	31

6.1	Eclipse Platform Core Plugins . . . . .	47
6.2	Bugzilla Report Classification . . . . .	48
B.1	Data Source Consolidation . . . . .	70
B.1	Data Source Consolidation . . . . .	71
B.1	Data Source Consolidation . . . . .	72
B.1	Data Source Consolidation . . . . .	73
D.1	Eclipse Platform Organization . . . . .	80

## List of Listings

3.1	CVS Log Example of a Later Revision . . . . .	13
3.2	CVS Log of the First Revision . . . . .	14
4.1	Processing Steps to Integrate Multiple Projects into the Target Schema . . . . .	22
4.2	Footer of a Mail Belonging to a Thread . . . . .	25
4.3	Processing Steps of the Communication Path Transformation . . . . .	27
4.4	Implementation of the getAssoziation Method within BugModelBuilder.java . . . . .	28
4.5	Implementation of the getMailPrefix Method within BugModelBuilder.java . . . . .	30
A.1	Function to get the Overall Contribution of the Revision's Author . . . . .	65
A.2	Function to get the Name of the Person Owning the Relating File . . . . .	66
A.3	Function to get the Overall Contribution of the Owner . . . . .	66
C.1	Test Queries . . . . .	76





# Introduction

## 1.1 Motivation

Since large and complex software projects consist of people playing different roles, the communication and collaboration among the team members are key success factors. This includes the discussion about user requirements or architecture and design issues, the communication with the users, and the contribution to the development. In a fast-moving world, the understanding of the system is crucial to the team members, especially during the realization phase, where the project faces high time pressure, and there is little time to obtain knowledge about unknown areas of the project.

In order to better understand the factors that influence the quality of software, Aberdour [Abe07] investigated Open Source Software (OSS) communities. He showed that the most important prerequisite for high-quality software is the understanding of the key areas of the communities. He introduced guidelines to achieve high-quality OSS, relying on code modularity, project management, and test process management. Furthermore, the team's culture and the environment are found to play an important role.

These findings motivate us to develop means and techniques that provide deeper insight into the project and its environment. As OSS communities are globally distributed, coordination is of high importance. Yet, being a virtual and informal community, no official structure exist. Therefore, the organizational structure can only be derived from the community members' behaviour. Since software changes over time, the history of a project provides valuable information on this behaviour. Such historic data is contained in different sources, such as diverse communication channels (mailing lists, developer forums, newsgroups, chats) and change management systems (code modification repositories, bug tracking databases, requirement tools). Combining different sources provides an integrated view on the project's development.

Having this integrated data does not yet provide insights, as it is too complex to interpret raw data. The human brain is more capable of processing visual information. This leads to the need for visual representation of the data in an intuitive manner, enabling the user's visual intelligence find the patterns of interest.

As one of the main goals is to observe the evolution of the project, the time dimension has to be taken into consideration, requiring dynamic visualizations, such as evolving networks.

## 1.2 Scope

In order to analyze the structures of OSS projects, communication and collaboration interactions are the main interest. Combining the two enables to build an interaction model, allowing for understanding the informal structures and dynamics within an OSS project. The data is not analyzed from a qualitative perspective, but the main goal is to provide a proper representation of the original data. This is achieved by integrating the different sources and modeling it with regard to the time dimension.

## 1.3 Objectives

This thesis aims at providing a representation to derive new valuable information about software development communities. Understanding the evolution of a project delivers new perspectives to analyze the situation.

The contribution of this thesis is to build an integrated data base containing cleaned data from modification reports, problem reports, and mail communication. Existing methods to extract, clean and integrate the data need to be evaluated and, if required, extended, or new methods need to be implemented. The data is obtained from the Eclipse Platform project.

A tool is implemented that visualizes the interactions within the project, considering the time-dimension. The social network graph is extended by adding quality features describing the different objects in the network (such as work packages). So far, no similar tool exists that allows discovering a project with such granularity and based on the multiple sources.

## 1.4 Outline

This thesis is structured as follows: Chapter 2 gives an overview of the research fields of social network analysis (including socio-technical oriented reports), software development and visualization techniques.

Chapter 3 presents two perspectives of different project members and outlines the motivation. We presents means and techniques to analyze the interactions of a software project team. In order to model this interaction, we will combine communication with collaboration information.

Chapter 4 outlines the data extraction and processing steps to obtain the integrated database. We first give an overview with a view to the integrated database. The detailed process steps are outlined and if necessary, the extensions, algorithms or implementations are described.

Chapter 5 presents the implementation of the Social Network Analysis Cockpit. First, we discuss the time and abstraction issue in order to approach the implementation concepts. The final section explains how to interpret the represented graphs.

The evaluation of the visualization and the case study of the Eclipse Platform Core project is outlined in 6. We describe the requirements for the project selection and give a brief overview of the core team. The aim of the following evaluation is to find patterns that allow to answer the questions outlined. This includes general patterns and roles we are interested in, such as *'Who are the key developers?'*, as well as scenarios that we try to spot.

Chapter 7 summarizes the thesis and draws the final conclusions. And we outline potential suggestions for future work.

The Appendices A - D contain detail information about used algorithms and describe parts of the implementation. Large overview pictures are placed in the Appendices.

# Related Work

This chapter gives a brief overview of the state-of-the-art in the research fields with respect to this thesis and outlines the delineations. The areas cover the social network analysis, the contribution to software development and this paper's main focus on the alliance of these two structures. The last section outlines current visualization techniques applicable to social networks.

## 2.1 Social Networks Analysis

The social network analysis has emerged in the sociology and exists in various forms and in many scientific disciplines. To gain information about the structure of Open Source Software (OSS) development systems, bug reports, mailing list archives, news groups or forums are extracted to model the actors' interactions. In general, all of the following quantitative studies faced the time consuming and exhausting effort of collecting and cleaning the data.

### 2.1.1 Communication Analysis

Crowston et al. [CWL<sup>+</sup>05] analyzed the coordination mechanisms in terms of work in OSS communities with a large number of developers and compared them to the corresponding rulings in proprietary development projects. The analysis is based on the Coordination Theory Approach framework, described in [Cro97] and [CO03], that models a group action in terms of actors performing interdependent tasks that require or create resources. The main problem they faced was to identify dependencies from only the messages of the three projects inquired, therefore the focus shifted on the observation of task assignments and shows that self-assignment was the most common one. The data explored were interactions between main developers extracted from the developer mailing list, and an online forum. The data had been mostly coded manually over several months. We will follow their notation of interaction for the rest of this thesis.

Howison et al. [HIC06] took a closer look at the dynamics of the social structure by undertaking social network analysis over time. They wanted to better understand how social structure in projects are changing over time. They inquired the average centralization over time, the change at the center and the stability of participation in project communications by modeling the network as presented by White [WBB76]. The study shows that the centralization scores are negatively correlated with the number of participants in the bug report discussions and that most of the participants are involved for only a small number of periods and according to the power law distribution a small number is involved for long periods. The examined data was extracted from

the SourceForge bug tracking system and the interaction coded between the sender and the immediately previous poster. This coding approach will later be evaluated in order to verify its significance. According to the conclusion they will extend these analyses with code repository contribution information for future research.

As outlined in the introduction, communication analysis covers only a part of the scope of this thesis whose aim is to consider all interactions occurring within a project team.

### 2.1.2 Topological Analysis

In [XGCM05] and [XCM05] Xu et al. examined multiple projects of different types, but on a higher level and in a static manner. They performed a quantitative analysis of Open Source Software developers by studying the development community at SourceFourge.net and classifying the actors in groups of project leaders, core developers, co-developers and active users. They showed that the development community is a self-organizing system besides the existence of the small-world phenomenon, the small average distance and the high clustering coefficient. The data extracted contains information about the roles of developers and the members' activities, such as bug reports, code submissions or forum discussions. The derived social network model is constructed by relating the developers to the project they contribute to or participate in. This again leads to relationships between projects where the resulting clusters corresponds to the projects properties. We will model the contribution information on a more detailed level and only for the visualization refer to their approach of abstraction.

## 2.2 Socio-Technical Networks Analysis

Ducheneaut [Duc05] analyzed the socialization of newcomers to the OSS community of Phython, showing that the integration of a new member is not only depending on his technical skills but also on his ability to learn how to participate and to build an identity for that his ideas will get accepted and integrated. He combines the social network built from the mailing list archive with the material structure based on CVS log data to look at the trajectories of participants from joining to contributing. To visualize the project's evolution he implemented the OSS Browser, which provides a dynamic view of the social network, built on the Conversation Map of Sack [Sac01], extended by the material view of source code entities that are connected to the corresponding committing developers. We will extend these entities by adding features like the ownership.

Sack et al. [SDD<sup>+</sup>06] continued this research field with an analysis across the three information spaces which build the socio-technical network; discussion, implementation and documentation. They tried to answer the questions how power is distributed, how links evolve between people and how the cognitive activity of discussions is influenced by the social and governance structures of the project. Mails, CVS logs and PEP documents of the Phython project served as data basis. Their current work is on analyzing the influence of the technical structure on the social structure of the discussion by correlating the structure of the implementation space (as it is incorporated in CVS logs and code dependencies) with the structure of the discussion space (as it manifests itself in the threads and quotations of the newsgroups and mailing lists of the project). The purpose is to show that the perspective that the 'ownership architecture' [BH98], containing information about who makes changes and extensions to which module of the system, and implying the 'Conway's Law' (developed and revealed in [Con68] and [HG99]) which says that the governance structure of the project has a direct influence on the structure of the software itself, can be inverted.

## 2.3 Software Development Teams

Software as being developed by people implies a dependency between the resulting system and the authors of the source code. This section introduces reports that examined software engineering and software evolution with focus on the participants or their roles.

### 2.3.1 Software Evolution

Version control systems like CVS [PtFSF98] contain information about the systems' history. Fischer et al. [FPG03] introduced an approach for populating a release history database that combined version data with bug tracking data. To obtain the release information that enable analysis on historic software sources, the EVOLIZERBASE is used and extended.

Girba et al [GKSD05] wanted to understand the interaction between different developers and the system. They proposed an approach to measure and define the ownership of a file to determine characteristics of developer behaviors. They process CVS log information to analyse the changes over time. The analysis are based on the ownership map visualization and shows different behavioral patterns. We will analyze the shortcoming of Girba's approach and propose a possible improvement.

### 2.3.2 Open Source Software

Aberdour [Abe07] addressed the question on how to achieve OSS quality by comparing best practices of OSS development with closed-source software development. He reported that high-quality OSS relies on having a large and sustainable community that has to be fully understood by the community members. The final guidelines to high-quality OSS imply high code modularity, rapid release cycles and many bug finders. His findings on quality justify our aim at providing means for a better understanding of the project environment.

## 2.4 Visualization

Network visualization is a well-researched field and there exists a lot of implementations and frameworks. Some applications provide the processing of external data, but few overcome the difficulties of visualizing evolving networks.

Many Eyes [IBM] is an online site, provided by IBM's Collaborative User Experience research group, to get insights into existing visualizations. To explore them, one can upload and illustrate the own data set. The published types of visualization vary from tree maps to bubble charts and support simple table data representations. For the visualization of multi dimensional data sets there exists no implementations.

Ogawa et al. [MO07] addressed the visualization techniques to enable the analysis of the evolution of the communication and collaboration activities. They extracted data from CVS repositories and mailing list archives. The visualization implementation is based on combining the repository view and the mailing list view. The repository is represented using the Windows Explorer tree visualization and the mailing lists are displayed as clusters within Sankey diagrams. The objective is to visualize large and evolving networks and to provide an insight into their details. The differentiation to this thesis is related to the use of a different approaches regarding the visualization of evolving networks.



# Interaction in Software Projects

This chapter outlines the motivation and presents the means and techniques to analyze the interactions of a software project team. The main focus is on the question about the inner life of the project, that consists of people playing different significant roles and of the products they develop. The collaborative interaction among the project members is reflected in the organization and has an influence on the project's outcome and its environment. The social structure of a community, based on communication, is combined with collaboration information representing working teams. This integration enables to further investigate the activities going on inside the project. The developed means and techniques are based on analyses of Open Source Software Communities, but they can be adapted to commercial projects as well.

The first section presents possible roles somebody can play within a project and outlines their perspectives. This is followed by a brief introduction in this thesis' main outcome. In order to understand interaction in software projects, the subsequent section explains how team members collaborate and communicate, and the final section presents techniques to model these interactions.

## 3.1 Perspectives

We concentrate on two perspectives, each based on a role within a project. Both perspectives are interested in similar questions, but from a different point of view. The first one is from the project leader's point of view, who wants to get information about the current status of the project. The second is the perspective of a newcomer, who needs to understand the organization and learn about the project's facts. In general, both are interested in who the community members are and what roles they hold.

Imagine you are a project manager, leading a software development project with parts of the implementation being outsourced. As the delivery deadline is coming closer and the quality testing discovers a higher number of defects than expected, you are presumably interested in the collaboration and coordination patterns between the different teams. To make another example, consider the leading developer is on leave for a longer period. Despite the deputy arrangement, it seems that the team does not harmonize and the deputy does not assume the leadership. A detailed view of the project's set-up would allow somebody to assess these situations.



### 3.1.1 Perspective: Project Manager

Important tasks of a project manager are the communication with the customer as well as with the team, and the planning and scheduling of the the team's activities. Knowing the current state of the project is a crucial factor. Maybe the team is grouped into several sub-groups or some are members of more than one group. The following questions are addressed to get an overview of the current situation and to identify the key and leading persons of a team.

- *Do sub-groups exist within the project team?*
- *Who is the key personality in respect of communication?*
- *Who is the leading person regarding code contribution?*
- *How are the working teams organized in terms of software development?*

Regarding the social structure of the community, a member communicating a lot more than others probably plays a decisive role in the project.

### 3.1.2 Perspective: Newcomer

Newcomers face the challenge to adapt themselves to the new environment before being able to contribute. The community with its members and their roles needs to be understood. The project's official set-up often differs from the informal organization structure. Whereas official responsibilities, competences and the project organization are usually accessible, the informal and situational structure are not. Yet they represent the actual network and therefore are of higher interest than the formal set-up. This situation based context can have an important meaning in a community.

Regarding the project organization, a newcomer is mainly interested in who are the key people in terms of technical tasks. To be able to answer questions like the following, allows a newcomer to contact the right colleagues. Besides the ownership of work packages, the knowledge of the different roles of the community members helps to get situated quickly.

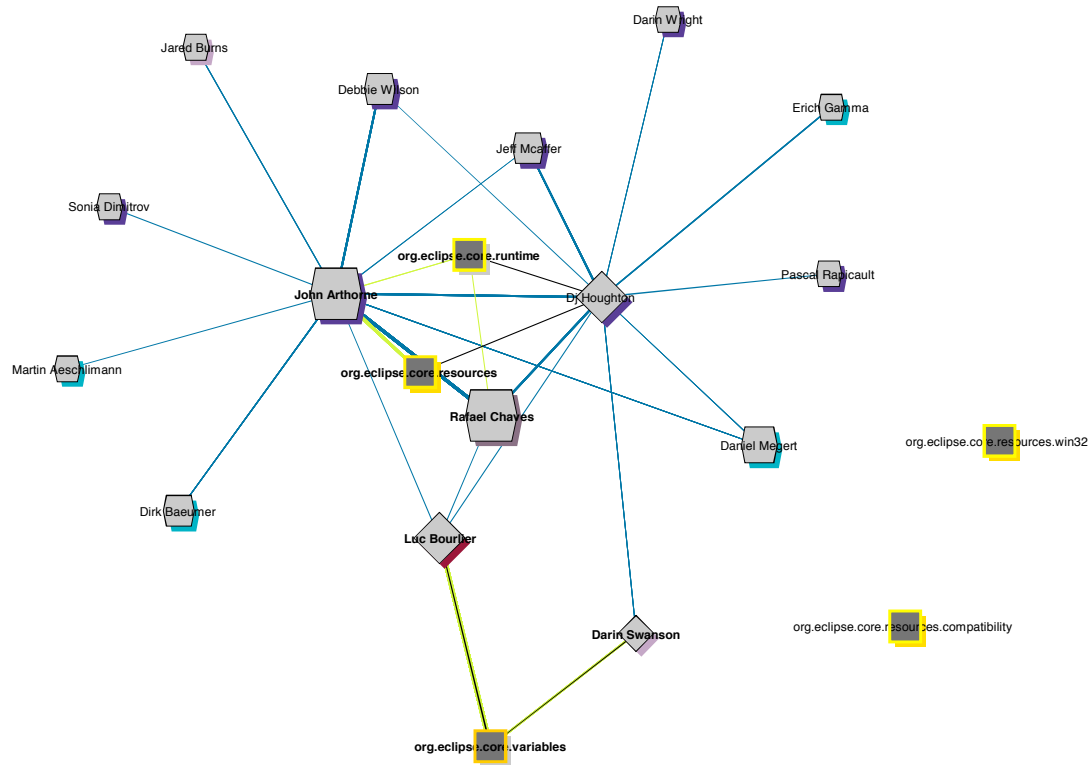
- *Who is a leading developer with the best general overview?*
- *Who is a key personality in terms of software development contribution?*
- *Who is working on which components?*

## 3.2 Social Network Analysis Cockpit

In order to explore the interactions occurring within a software project, we integrate data from mailing lists, Bugzilla databases and CVS repositories in a social network graph, and present a visualization tool.

Figure 3.1 shows a sample view of an interaction network. The nodes represent project members and work packages and the edges illustrate the communication between people or the contribution of a developer to a work package. In order to represent the aspects introduced in the previous section, we make use of various graphical designs such as different shapes for nodes or different colors for edges. The size of a node represents a project member, for example denotes his communication activity.





**Figure 3.1:** Interaction Network

Consider the following example: The two leading persons John Arthorne and DJ Houghton can be identified at first sight. Gladwell explains in *The Tipping Point* [Gla02] behavioral phenomena and patterns (in terms of psychology, sociology and epidemiology) like *the Law of the Few*. It says that in order for something to become heard, the right information needs to get channeled through the right people. J.A and D.H. are such type of *Connectors* who link different networks, while being the central communication person within each of the sub-networks. The chance of getting the right answer requires addressing the right people - like one of these connectors. What happens to the networks' structure when one of these two persons is absent or permanently leaving the project?

The aim of this visualization is to enable an understandable perception of the project's set-up and to illustrate its dynamics to explore the evolution interactively. In order to examine specific periods of the project's history, we take the time into consideration. This time shifting enables to move forward and backward within the history and to survey and analyze the evolution of the social network, which is displayed by animating the change of the graph. Details of the visualization and the interpretation of our social network graphs are outlined in Chapter 5.

## 3.3 Understanding Project Collaboration

To understand the collaboration within a project, the used instruments are of interest. How do project members work together, share information and knowledge? What are the communication channels they use and for what purpose? This section concentrates on common instruments within open source software projects.

### 3.3.1 Software Development

Since software is changing over time, modifications on source components are tracked. The information provided by CVS [PtFSF98] is evaluated to learn something about the collaboration of a development team. Within this thesis the features of the CVS are sufficient to extract the required information. The content of the source code files is not included.

#### Concurrent Version System

In order to get insights into the project's collaboration, we are interested in the composition of the working groups and in questions like who is working on the same source code modules. Therefore we need to know which developer made a modification to what file, to which extent and at what point in time. The log contains the number of lines added or deleted within one CVS operation. This provides the basis to derive the amount of modifications a developer has made, to calculate the contribution made to that file so far, and to determine who is the owner of the file at that respective time. Consequently we know if the committing person is also the owner of the file and if not, if he possibly takes over ownership. Furthermore the commit message contains information on the intention of a modification. In case of a bug fix revision, the commit message (additional to the described purpose of the modification) may also contain the bug number.

### 3.3.2 Communication Channels

Knowing the communication pattern helps to understand the organization of a community. In a virtual world, where the members of an open source software development team are globally distributed, the information is exchanged across different channels. This requires systems and tools that help to organize the software change process and management tasks.

#### Mailing Lists

A project team needs tools to get and stay organized and to communicate with each other. To exchange information in open source software communities, where people have separated work places, mailing lists are a common instrument. Chat tools or newsgroups are other communication channels, but relevant information regarding software development issues should not be exchanged over these channels, mainly because the information is not stored and therefore not accessible to other team members afterwards. At eclipse.org [Inc01] for example mailing lists are intended for use by developers and users to discuss design and implementation issues.

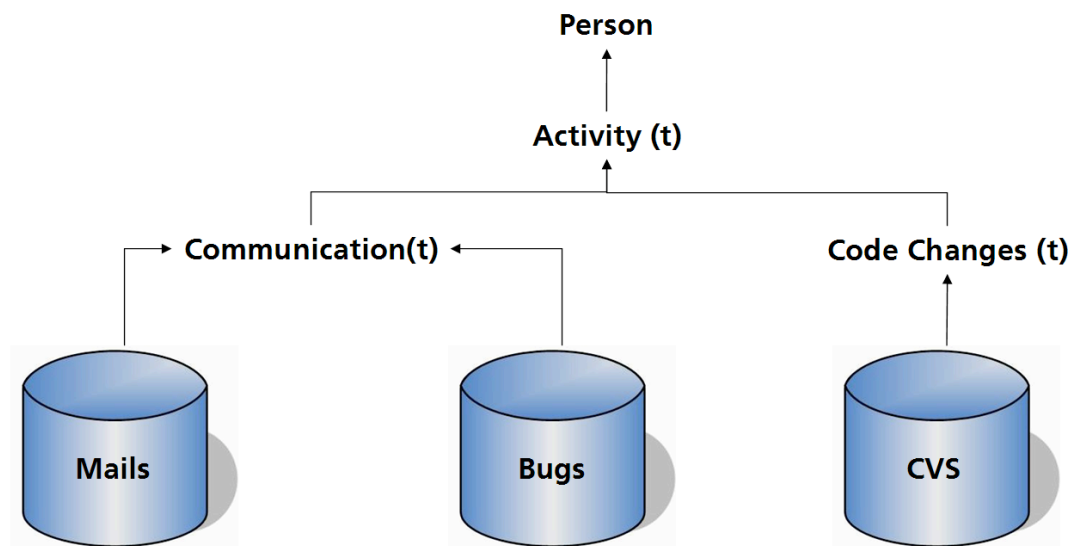
#### Problem Reporting Tools

Communication concerning software development tasks happens not only in mail traffic, but also in other tools used within the software development process. For example problem reporting tools, where tasks can be reported, assigned, commented and traced. We assume that tools like

Bugzilla [Fou98] are the most frequent used communication platforms regarding software development and that discussions among the developers take place there. Discussions about general questions or communication with outsiders (for example users) are covered by mailing lists.

## 3.4 Interaction Target Model

The interaction target model combines the three data sources; the revision information, the mailing lists, and the problem reports. This permits an integrated view on the project's set-up.



**Figure 3.2:** High Level Integration Concept

Figure 3.2 shows the high level integration scheme that includes all three sources. Applying Crowston's [Cro97] process mapping concept, where a process is described as '*a sequence of activities performed by organizational actors that produce and consume resources*', an **actor** is a real person in the role of a community member, an **activity** either a source code modification, the posting of a mail or a contribution to problem reports at a given point in time and a **resource** a file or a task arising from discussions.

CVS repositories have knowledge about the history of a file, but file specific information like the actual file size is not kept. Hence, the raw CVS log data is extended by deriving features characterizing the behaviour of developers working on the same piece of code. Figure 3.3 shows the unified overview of the data required to answer the questions introduced in the first section of this chapter. The bordered areas indicate the two parts underlying the networks. The yellow section includes the subjects required to model the author-source component network of the system, the blue one implies the communication subjects underlying the social network.

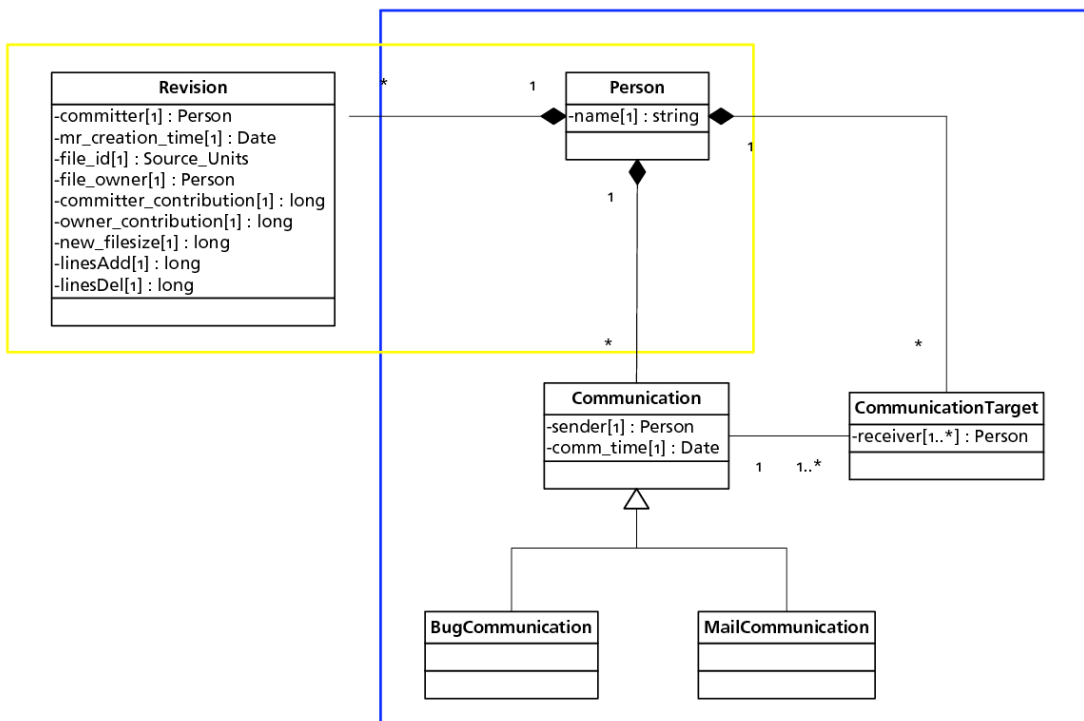


Figure 3.3: High Level Interaction Model

### 3.4.1 Software Evolution Model

To model the structure of a work team, we need to know who is or was working on which component of the system. The CVS log provides details about code revisions that enable to derive these allocations. The minimal information required is the author of the modification (including creation or deletion) and the affected file (including path and file name). For each revision the time stamp of the CVS commit, the corresponding commit message and the extent of the file modification (number of lines added and deleted) are needed and provided by the same log. In order to explore different roles and patterns, we are interested in the ownership of a file. This enables to analyze the interaction between the developer and the owner of a file and in particular, how the communication between the two proceeds. Furthermore, the ownership incorporates a special role within the project team. By distinguishing the roles of the team members, this enables to state the project more precisely.

#### Source Code Ownership

Besides the author of the modification, we are interested in who is the owner of a source file. In order to determine this ownership, the approach of Girba et al. [GKSD05] is analyzed. Girba proposes a measurement for the notion of code ownership by evaluating the CVS log. His aim was to provide a solution that gives fast results and needs no deeper analysis of the code content. He defines the owner of a piece of code as being the developer that owns the most lines of it. The total file size is defined as the sum of the deltas that result from the calculation of the number of lines added, minus the number of lines deleted. The contribution of a developer corresponds to

the sum of the deltas of the revisions that are committed by this developer accordingly.

To make an example, consider Listing 3.1. The delta of the number of changed lines for this revision is 5 ( $11 - 6 = 5$ ). The author is *johna* who is also the author of the revisions number 1.3 and 1.4 of this file. His contribution up to revision 1.6 is six lines. The total file size at that point in time is 207 lines of which six are owned by *johna*.

```
Revision 1.6 - (view) (download) (annotate) - [select for diffs]
Fri May 17 19:38:00 2002 UTC (4 years, 10 months ago) by johna
Branch: MAIN
CVS Tags: R2_0, Root_Bug_21029, Root_JA_AliasSupport_20021128,
        Root_JA_Mounting_20020924, Root_dj_20020718, r201_v20020801,
        v20020521, v20020521a, v20020528, v20020528a, v20020529, v20020530,
        v20020531, v20020601
Branch point for: Bug_21029, JA_AliasSupport_20021128,
        JA_Mounting_20020924, R2_0_1, ResourceMapping_20020712, dj_20020718
Changes since 1.5: +11 -6 lines
Diff to previous 1.5
Added NLS tags and updated copyright notices
```

**Listing 3.1:** CVS Log Example of a Later Revision

The CVS log does not provide any information about the file size of the first revision. Listing 3.2 shows that the log of a first revision does not contain the line '*Diff to previous x.x*'. Girba does not take any initial file size into consideration because the CVS log does not provide it. The aim of this thesis is to analyze the communication between the committing developer and the owner of the modified file. There is the risk that due to the missing consideration of the initial file size, Girba's approach is not sufficient to determine the file owner or does result in misleading changes of owner.

To make an example, consider a file with three revisions. The initial file size is 123 and the delta (resulting from the difference between lines added and deleted) is zero for revision 1.1 and five and seven for the following revisions. With Girba's approach, the author of the third revision takes over the ownership. If the author of the first revision does not equal the third one and the initial file size of 123 had been considered, the commit of the third revision would not lead to a change of ownership.

We assume that the initial file size ought to be considered. Therefore the approach including the initial file size is tested against Girba's approach. In order to get the initial file size, the lines of each initial revision 1.1 are counted and processed as number of lines added. The detailed calculations and the processing are described in Appendix A.

For comparing our approach with the approach of Girba we perform an experiment with plugins of the Eclipse project. We compare the occurrence of CVS commits over a period of nearly five years. The results are listed in the following Table 3.1. Both approaches score over 50% regarding the alien commits.<sup>1</sup> This means that approximately every second commit is done by somebody who is not the owner of the modified file. The frequency of a CVS commit performed by a person not owning the file is comparable between the two approaches, whereas a striking difference arises in the extent of changes of owner. Regarding the changes of owner, Girba's approach leads to 425 changes of owner whereas with our approach only 51 of the 8583 revisions

<sup>1</sup>A commit done by an author not being the owner of the relating file is called an alien commit.

```
Revision 1.1 - (view) (download) (annotate) - [select for diffs]
Wed May 2 17:14:00 2001 UTC (5 years, 11 months ago) by dj
Branch: MAIN
CVS Tags: v102, v103, v104, v105, v106, v107, v108, v108a, v110, v112
0.102
```

**Listing 3.2:** CVS Log of the First Revision

result in a change of ownership. Without considering the file size of the first revision, a change of owner happens seven times more often. This frequency seems to be substantial, but by surveying the dates with numerous commits, where the two approaches result in different values, a closer look at the commit messages of these revisions shows an interesting pattern.

Approach	Girba	%	Own Approach	%
Amount of Alien <sup>1</sup> Commits	4355	51%	4634	54%
Amount of Changes of Owner	425	5%	51	0.06%
Number of Revisions	8583	-	8583	-

**Table 3.1:** Comparison with Girba's Approach

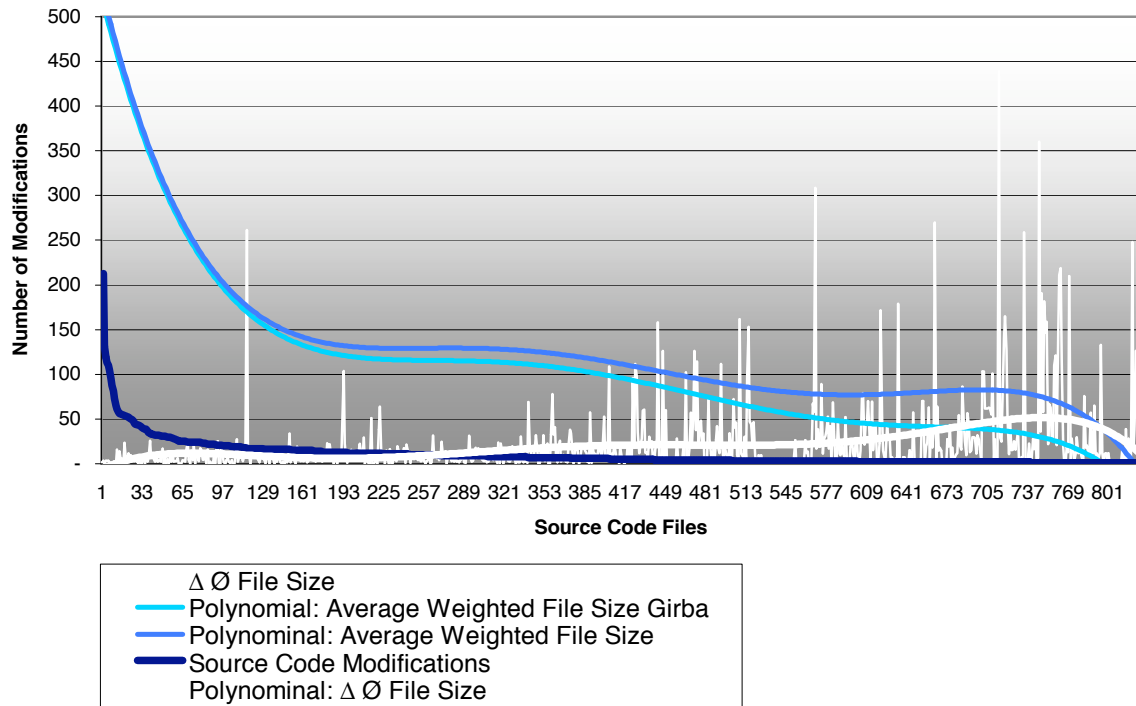
Analyzing the commit messages of revisions which result in a change of owner using Girba's calculation, the modification does not seem to rely on changing the semantics of the file, but on just reengineering the content. The following examples list the six most frequent messages, but represent a quarter of all revisions resulting in a change of owner with Girba's approach.

- Added NLS tags and updated copyright notices
- Code format.
- update javadoc with obsolete and new disclaimers
- Reformat Equinox code according to the Core formatter settings
- Add copyright notice
- \*\*\* empty log message \*\*\*

Due to the missing initial file size, smaller delta of changed lines enforces a change of owner, compared to the approach including the amount of lines of the first revision. The following Figure 3.4 shows that the more often a file is changed, the less the calculated file size depends on the underlying approach. This meets the expectation, that the initial file size only influences the ownership question in case of files being modified rarely over time.

The dark blue thick line indicates the distribution of the modification frequency of all files. The file that has been changed most frequently counts 284 revisions, whereas for 88 of the 826 files the actual revision is still the first committed ever. The white color illustrates the variation of the file size of the two *file ownership* approaches with the thick line representing its polynomial trend line. The two lighter blue lines show the polynomial trend line of the average file size of each approach (with the lighter line indicating Girba's concept). In addition, this graph shows that the more often a file is changed, the larger its size gets.

According to the findings discussed in this section, we will follow our approach that considers the initial file size to determine the code ownership.



**Figure 3.4:** Correlation between the File Size and the Frequency of Modification

### 3.4.2 Communication Model

Communication within a project community takes place across different channels. A special characteristic of an open source software community is the distribution in terms of location. Compared to a commercial team, the members of an open source software project are distributed all over the world, forming a virtual team. They need organizational instruments to share and exchange information. Regarding the control and assignment of tasks, they use appropriate tools. Internet mailing lists are instruments to address information to a dynamically changing community. A mailing list has a list of subscribers receiving the messages processed by the reflector address. Another common tool is Bugzilla, used to manage problem reports. We assume, that most of the core developers of the community interact using such designated tools. This section shows how to model the communication path between sender and receiver of a message, referring to mailing lists and Bugzilla reports.

#### Deriving Communication Paths from Mail Traffic

Communication happens between a sender and at least one receiver at a certain point in time with a given content. To specify the social structure of an open source software community, we derive communication paths of mail traffic and problem reports.

Discussions arising from an initial mail can be grouped as threads. Mails referring to the same subject are kept together. Within mailing list threads, the messages can grow in a dendritic way. The following sample explains how communication paths can be modeled from threads.

Figure 3.5 shows a thread from an online mailing list archive. A mail addressed to a mailing list is processed by the reflector and sent to all subscribers. This means that the *To:* address is always the mailing list address. The identification of the sender is given by the *From:* field, but



- [\[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Thomas Watson
  - [RE: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Philippe Ombredanne
    - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Bob Foster
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Pascal Rapicault
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Bob Foster
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Jim des Rivieres
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), John Arthorne
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Randy Hudson
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Jeff McAffer
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Randy Hudson
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Jeff McAffer
      - [Re: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Bob Foster
  - <Possible follow-ups>
  - [RE: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Ed Burnette
    - [RE: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Jeff McAffer
    - [RE: \[platform-core-dev\] Bundles which do not require org.eclipse.osgi](#), Philippe Ombredanne

Figure 3.5: A Mail Thread of an Online Archive

knowing the sender of a message is not sufficient to model a communication path between two persons. Information about the receiver needs to be acquired from subsequent answering mails.

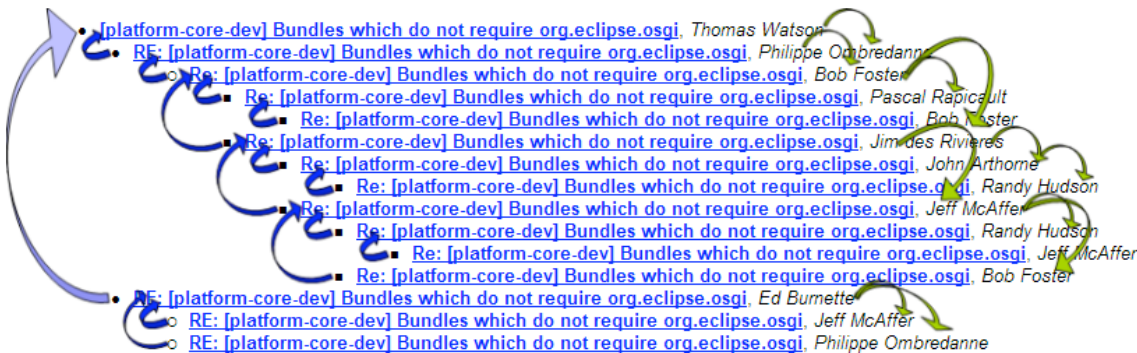


Figure 3.6: The Adjusted Mail Thread with the Corresponding Communication Paths

We consider the *From*: address as the sender of a message and the *To*:’s and *Cc*:’s as the receivers. Figure 3.6 illustrates the tree of the example thread. The derived communication paths are indicated by the arrows. A blue arrow represents a *To*: path, the green a *Cc*:’s. The *To*: receiver implies the sender of the mail somebody answers. So we treat the sender of the first message as the receiver of the second and answering message, although all subscribers to the mailing list get the mail. To derive *Cc*: receivers we consider the person answering a mail as an intended receiver of the previous mail, but only *Cc*:. In case this person is already the *To*: receiver (as it applies with the mails number 3 to 5 between Bob Foster and Pascal Rapicault) no additional path is derived, because we assume that a mail is not sent to a person twice. The examined sample thread consists of 15 mails, all sent to the mailing list address, which results in 25 communication paths.

## Deriving Communication Paths from Problem Reports

The second source outlined for modeling communication paths is a problem reporting tool such as Bugzilla, where members can create reports and comments and give answers to former editors or commentators. Within a Bugzilla problem report, a person can occupy the role of the reporter that opens and describes the report, of a person registered as *Cc*:, of the assignee that overtakes the



ownership or current responsibility, or of a commentator that writes an explanation or annotates the report.

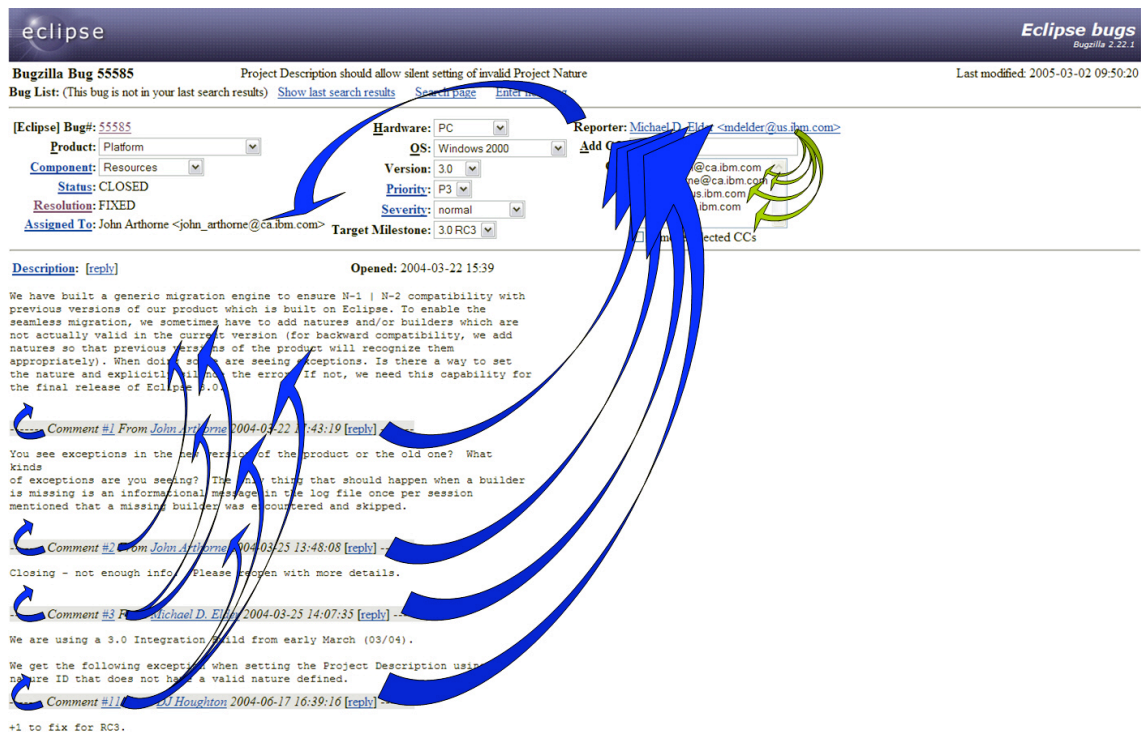


Figure 3.7: Communication within Problem Reports of Bugzilla

Regarding the comments on a report, we assume that a person dealing with a problem report, has complete knowledge about the previous written comments or activities. This approach differs to the one used within mail threads where no integrated view over the actual mail thread exists and a mail is directly addressed to someone. Within a problem report the communication paths are derived using other mechanisms. Figure 3.7 illustrates the approach by a precise report example.

Again communication consists of a sender, at least one receiver, a time stamp and the subject and/or the content of the message. The transition of problem reports to communication objects and targets is more complex than the conversion of mails.

The list below specifies the actions within problem reports where communication occurs and indicates their resulting communication paths (whereas an  $\rightarrow$  means 'communicates with').

#### Creation of a Problem Report (at a given point in time)

- Reporter  $\rightarrow$  Assignee
- Reporter  $\rightarrow \forall$  Cc:
- Content: Description

#### Adding a Comment (at a given point in time)

- Commentator  $\rightarrow$  Assignee
- Commentator  $\rightarrow$  Reporter
- Commentator  $\rightarrow \forall$  previous Commentator
- Content: Comment

There are two different actions: creating a report and writing a comment. The communication emerging from report creation is the assignment of the task to the assignee by the reporter and the notification of the persons registered as Cc:. As long as the report is in an unresolved state, the following comments result in further communication. Each commentator addresses their comment to the reporter, the assignee and all former commentators. This approach differs to the one of Howison et al. [HIC06], where only a communication to the immediate previous poster was assumed. Content analysis of problem reports show, that the comment often refers not only to the precedent comment. Regarding communication with Cc: addressees we assume, that if somebody is concerned he or she will get involved as a commentator (see comment # 11 from DJ Houghton). Therefore we do not map the communication to Cc: after the report creation. To simplify matters and because we assume that historical reports are in a stable state we do not consider the bug activities.

The illustrated example consists of 15 comments (including the first one) and results in 36 communication paths, including three Cc:’s. Compared to the mailings it is expected that data from problem reports result in more communication information.

## 3.5 Summary

In this chapter we have presented the essential concept of this thesis. Section 3.1 outlines the research questions and gives an idea on the aspects by viewing at two roles that a member of a project team can play. The following Section 3.2 introduces the implemented Social Network Analysis Cockpit that enables to explore the history of a project interactively. Section 3.3 describes what is considered to be project collaboration within this thesis. The idea is to combine mailing lists and Buzilla data with collaboration information gained from CVS repositories. This enables an integrated view on the projects’ set-up with the aim at better understanding the project. Section 3.4 outlines the modeling and integration of the data sources. The software evolution model has been extended by ownership information. The underlying approach, introduced by Girba et al. has been discussed and improved. Section 3.4.2 models the communication paths that are derived from mailing lists and from problem reports.

The following Chapter 4 describes the data processing steps that are required to gain the integrated database, containing mails, Bugzilla reports and CVS revision information. This includes the extraction, cleaning, formatting and integration of the three data sources. These process steps are needed to enable the design and evaluation of the visualization implementation following in Chapter 5.

# Data Preparation

This chapter outlines the data extraction and processing steps to gather the integrated database. We describe the methods and tools of existing implementations and we discuss possible improvements. As we make similar experiences like Crowston and Howison [JH04], details and hints regarding the implementation of data extraction applications can be found in their publication. The focus of this chapter is to present the handling of the challenges focused while integrating different data sources.

First, we give an overview of the data processing with a view to the integrated database. For each of the data sources, the detailed process steps are outlined, and if necessary, the extensions, algorithms or implementations are described. The chapter is finalized by addressing the integration.

## 4.1 Data Processing

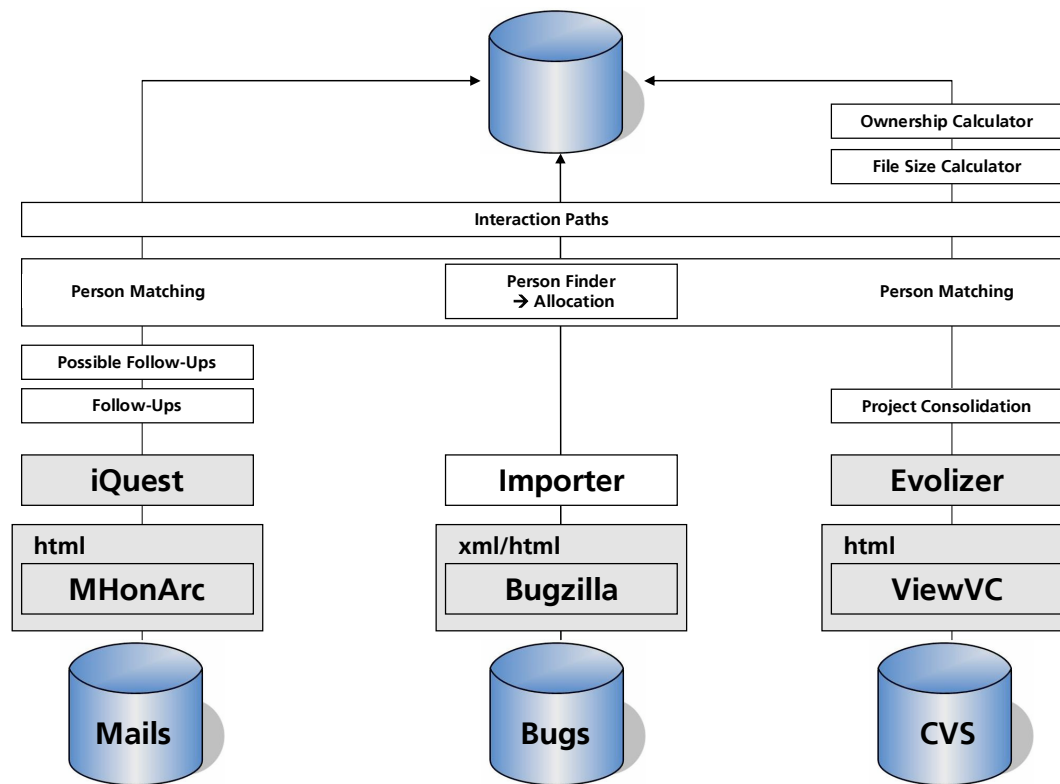
The data preparation process can be separated into several stages each finalized by a cleaning and quality control task. It includes the extraction and the integration of the different data sources. In a first step, the selected data is parsed and stored, for example in a database. In an additional step, the data is extended and processed, followed by the integration of the varying sources.

Figure 4.1 shows the overall process model. The source data is either stored within a running system as it applies to CVS repositories and bugzilla databases or archived as it applies to mailing lists. Since the original data is not accessible directly from its stored location and not provided by the hosting authority, the data needs to be extracted from an online view. These online representations are generated by tools. The applications running on the source system are illustrated by the three larger grey boxes at the bottom of Figure 4.1.

The simplicity of parsing the data from these online views depends on the representation, namely on its underlying tool, the information coding, and the format. The most frequent provided format is HTML, as it is common for internet sites. Depending on the tool, XML is supported as well. The three boxes right above the data source illustrate the data extraction layer, while the grey color marks external tools.<sup>1</sup> The superordinate layers include extension and integration tasks. The next layer addresses the mapping of person information. This is one of the main issues regarding the integration of the three different data sources, as it applies to data sets containing overlapping information. The superordinate derivation of the interaction paths follows the person mapping. Both of the last mentioned layers span all data tracks. Finally, there are extension components like the file size or ownership calculator.

---

<sup>1</sup>The white boxes depict components implemented within this thesis.



**Figure 4.1:** Data Source Integration Process

Figure 4.1 illustrates only the process components which include extraction and extension algorithms. Underlying data meta models are not shown in Figure 4.1, but can be found in Appendix B.1. The following section outlines the processing for each of the three data sources in more detail.

## 4.2 Data Extraction Tools

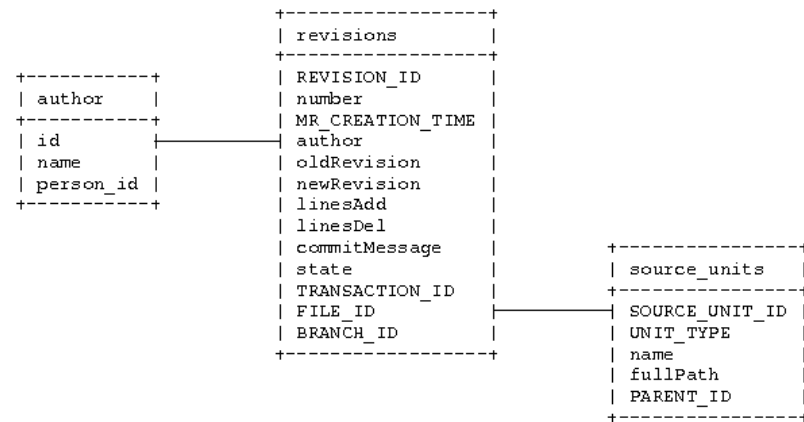
Since there is no existing data source available that covers the requirements of integrating all data contained in the interaction model, the data has to be extracted from its source system. The following sections outline the applications and qualify their capabilities and limitations.

### 4.2.1 Evolizer CVS Importer

The Evolizer is an integrated tool platform for software architecture and evolution analysis developed by the Software Evolution and Architecture Lab of the Department of Informatics of the University of Zurich [SEA04]. It is built on top of the Eclipse IDE and provides a set of meta models and corresponding data model extraction tools. The Evolizer CVS Importer is dedicated to importing CVS log information from a CVS repository, building the object model and storing it into a SQL database using Hibernate [Inc]. The Evolizer Plug-in is used to import the CVS

log information required for our analyses.<sup>2</sup> As the meta model covers a broader scope than required, the following list shows the processed classes and tables. Figure 4.2 shows the original data schema of the Evolizer. The CVS log extraction track is indicated by the third column in Figure 4.1.

- Author
- Revision
- Source Unit



**Figure 4.2:** Evolizer CVS Importer: Versioning Model

## Restrictions

The Evolizer provides a parser algorithm for CVS logs, as well as a meta model to store the extracted data in a database. The Evolizer CVS Importer Eclipse Plug-in gains access to the CVS repository of a project configured within the Eclipse workspace. The current implementation builds an object model for each project for which the Evolizer Importer task is executed. The execution task supports the processing of only one project at once. This implies that the scope of the created objects is one specific project and that an object contained in more than one project is created n-times and without any relating connections in between. Therefore the intended use of the tool is restricted to a scope, where an object cannot be contained in more than one project or the impact of this restriction can be disregarded.

The extraction of consistent data for a large software project being developed using the Eclipse Platform is a supported action. Today a lot of projects are organized in a modular way, namely that a set of Eclipse projects build a system. The Evolizer's meta model needs to be extended to enable the extraction of several projects and to create a consistent object model spanning these multiple projects. For the revision and source unit information, there is no issue. The partitioning of the source units and their revisions is given by the modularization. While a file or source unit is assumed to exist only once, an author of a piece of code can work within different projects. Importing several projects developed by the same team, leads to the creation of each author for several times. The used Evolizer implementation does not distinguish between projects. In order to prevent the data source from unnecessary redundancy and missing object relationships, the

<sup>2</sup>Revision 230 as of 2006-06-23.

project information has to be added. The simplest way to do this, is to extend all objects with the project information.

In case of multiple projects and the need of reading the data from the database after the import, another problem arises. The object keys set by the import are increasing integers. They are generated automatically, starting from 1 within each project and object type. Including the project information as an object attribute is an additional advantage to resolve this issue.

## Project Consolidation

Each project is processed as a single Evolizer Importer task and stored as separate schema within a MySQL database.<sup>3</sup> The schemata include the complete Evolizer CVS data model. In order to enable the import of multiple projects, a table containing the project's name and id is created (*util.plugin*). The target schema is called *cvs\_src* and includes the required tables of the Evolizer CVS model (see Listing 4.2.1 above). While copying the source tables to the target scheme, each table is extended by the project's id. Appendix 4.1 shows the detailed processing.

```

1  INSERT INTO util.plugin(name, product_id, domain_id) VALUES ('org.eclipse.core.
    variables', 1, 1);
2  SELECT id FROM util.plugin WHERE name = 'org.eclipse.core.variables';
3
4  INSERT INTO cvs_src.author SELECT *, 17 FROM cvs_core_variables.author;
5  INSERT INTO cvs_src.releases SELECT *, 17 FROM cvs_core_variables.releases;
6  INSERT INTO cvs_src.releases_revisions SELECT *, 17 FROM cvs_core_variables.
    releases_revisions;
7  INSERT INTO cvs_src.revisions SELECT *, 17 FROM cvs_core_variables.revisions;
8  INSERT INTO cvs_src.source_units SELECT *, 17 FROM cvs_core_variables.
    source_units;
9  INSERT INTO cvs_src.transactions SELECT *, 17 FROM cvs_core_variables.
    transactions;

```

**Listing 4.1:** Processing Steps to Integrate Multiple Projects into the Target Schema

Before the tables from the different import schemes can be copied to the target schema, the *cvs\_src* schema and its designated tables have to be created. First the project is registered in the *util.plugin* table to obtain a unique key for the project (indicated by the operations of line one and two of the Listing 4.1). The following transactions copy the original data extended by the project key to the target schema. Note that there are two ways of creating the tables of the target schema. Either the tables are copied within the first project's processing and the subsequent projects are added or the tables are created using the `CREATE TABLE tbl_name LIKE old_tbl_name` command. The second option keeps possible structural characteristics like primary keys and indices.

## Data Enrichment

The original Evolizer `org.evolizer.base` implementation has been modified and extended. The modification affects the CVS log information parsing. As shown in Section 3.4.1, the initial file size has to be considered. While parsing the CVS log of a file's first revision, a file parser component reads the file and counts the number of lines. According to this, the attribute *linesAdd* of every 1.1 revision indicates the total number of lines for this file.

<sup>3</sup>The raw data can be found on the enclosed CD.



To get the number of lines of the first revision, the package `org.evolizer.base.versioning.cvs.parser` has been modified.<sup>4</sup>

- **CVSParser.java**
  - Added method `getLinesOfFile(String revNr)` to get the file size (of the first revision)
- **SourceCodeFileParser.java**
  - New implemented class to read the number of lines of a file's revision

In an additional step, the ownership information is derived. This follows after parsing the complete data, because the history of a file underlies this derivation. The consolidated schema `cvs_src` does not yet contain this information. The attributes indicating the owner of a file and the boolean ownership flag are included in the final *report* schema (see Section A for the algorithms). Another derived information is the bug number contained in some of the commit messages that indicates the revision as being modified to fix a bug. This information is extracted and stored in a separate attribute. It contains the bug number and relates this revision to the corresponding problem report.

## 4.2.2 iQuest Mailing List Parser

Compared to CVS repositories and Bugzilla where the data is stored in databases, mailing lists do not support saving the exchanged mails. Only mailing list archives provides access to those mails. They are a collection of past messages and often include searching and indexing functionality. At `eclipse.org` the mailing lists are archived using MHonArc, a free software program to convert mails to HTML views [MHo94]. The first track in Figure 4.1 shows the extraction processing for mailing list archives.

iQuest [Glo] is the commercial version of TeCFlow, a set of tools to visualize the temporal evolution of communication patterns among groups of people. It contains a component to parse mailing lists and import them into a MySQL database. In order to parse the mailing list archives at `eclipse.org`, the iQuest's `OnLine.jar` component is used.<sup>5</sup> The details about the underlying schema and the iQuest processing follow in the next section.

## Data Processing

The following enumeration 4.2.2 lists the process steps of the mailing list archive import with the iQuest component `OnLine.jar`. Mailing list archives consist of several `.html` pages listing the mails in form of threads. The `OnLine.jar` application parses all mails whose link is contained in the specified page. Each page is processed separately and results in an own dataset. Step number four is required because the import creates an own dataset for each parsed `.html` page. To merge several datasets iQuest provides an according functionality. Step number ten is only required if the data is processed within iQuest.

1. Execute `OnLine.jar`: Parse mailing list online pages
2. Stop MySQL Service
3. Copy the MySQL schema *test* and rename it (to for example *mails*)
4. Merge the datasets with iQuest
5. `DELETE FROM mails.comm_target WHERE tag = 'Cc';`
6. `CREATE TABLE logging;`
7. `SELECT mails.extract_followups();`<sup>6</sup>

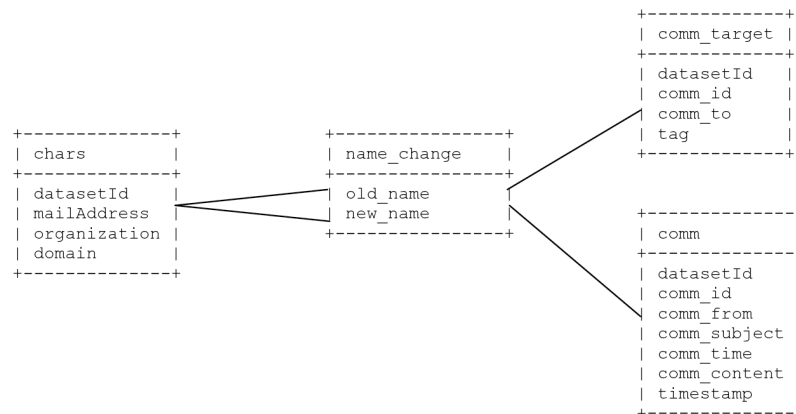
<sup>4</sup>The modified source code can be found on the enclosed CD.

<sup>5</sup>The iQuest components including the `OnLine.jar` to parse mailing lists can be found on the enclosed CD.

<sup>6</sup>The implemented MySQL functions can be found on the enclosed CD.

8. `SELECT mails.extract_possibleFollowups();`<sup>6</sup>
9. `SELECT mails.clean_content();`
10. Reference and follow-up authors mapping with iQuest → fill the table *name\_change*

## iQuest Schema



**Figure 4.3:** iQuest Mail Communication Schema

Figure 4.3 shows the iQuest schema to store the mails. The table *chars* contains the person information, with the *mailAddress* being either a person’s mail address (in case of the person being the sender of the mail) or the name of a person (in case of the person being the addressee of the mail). Since the dissolving of the mails’ addressees does not result in the entry of the receiver’s mail address, the To and From persons have to be mapped, as they can be in both positions. This mapping is contained in the table *name\_change*. The mapping is created during the visualization processing within the iQuest tool. Within the data preparation for this thesis, the table *chars* is replaced by the integrated table *person* and the table *name\_change* is dropped completely. To process the social network with the iQuest, these tables remain required.

## Extensions

Importing mailing list archives with the ordinary iQuest processing does not result in many communication paths. The addressee of a mail sent to a mailing list is always the mailing list’s address and the subscribers to the mailing list are not known. Since the mailing list parser algorithm of iQuest sets the mailing list’s address as the receiver of a mail, the resulting communication path (tagged with Cc:) relates the sender to the mailing list. Due to this thesis’ focus on the relationships between people, all iQuest Cc: entries containing the mailing list address are deleted. It is assumed, that threads contain additional information to derive communication paths with a higher entropy.

Since the mails are not extracted from the source directly, there is no indicated reference between mails within a thread. As described in Section 3.4.2, a mail thread contains information about subsequent mails. The iQuest import algorithm does not consider this information. To gain these additional communication paths, the amount of the parsed mail content is extended to include the (possible) Follow-ups (see Figure 3.5 and Figure 3.6). The process steps are outlined in Appendix 4.2.2.



The mail representation provided by the MHonArc needs to be considered to extract the Follow-ups.<sup>7</sup> The imported content of a mail starts with the message body and in addition contains the .html page footer. The *References* are set by the importer algorithm of iQuest. The MySQL functions number seven and eight of Listing 4.2.2 derive more communication paths by examining the page footer.

```
Follow-Ups:
Re: [platform-core-dev] Bundles which do not require org.eclipse.osgi
From: Bob Foster
References:
[platform-core-dev] Bundles which do not require org.eclipse.osgi
From: Thomas Watson
```

**Listing 4.2:** Footer of a Mail Belonging to a Thread

Listing 4.2 shows an example of a message footer. If the footer contains the text *Follow-Ups*;, the function 7 creates a communication path between this mail's composer and the person following the *From*: tag. Function 8 extracts possible Follow-ups by analyzing mails with the same subject and not yet considered.

### 4.2.3 Bugzilla Parser

The Evolizer provides a meta data model and an importing algorithm for problem reports. The importer requires a revision's commit message containing a bug number. It is assumed, that the communication of the core development team covers a broader scope of problem reports than based on bugs whose number is contained in commit message. According to the colored section in Table D.1, the corresponding bugs are based on their classification in terms of *Product* and *Component*. By querying the Bugzilla database the according bug list is extracted.<sup>8</sup>

The `org.seal.snanalyzer` Eclipse Plug-in provides an importer functionality to parse a list of bugs.<sup>9</sup> The underlying meta model is the `org.evolizer.model.issuetracking` model.<sup>10</sup>

#### Problem Report Data Model

The Evolizer `org.evolizer.model.issuetracking` data model is modified to match the requirements. The modifications are listed below and affect mainly the entity classes<sup>11</sup>.

- **EvolizerIssueTrackingModelProvider.java**
  - Added annotated class MailAddress.class
- **Comment.java**
  - Changed the persistence property of the comment attribute to @Lob
- **Issue.java**
  - Extended Resolution with REMIND and LATER

<sup>7</sup>See Section 4.2.2 for more details about MHonArc.

<sup>8</sup>The imported bug list (buglist.eclipse.platform.core.csv) can be found on the enclosed CD.

<sup>9</sup>It has been implemented within this thesis. The source code can be found on the enclosed CD.

<sup>10</sup>Revision 336 as of 2007-01-10.

<sup>11</sup>The modified source code can be found on the enclosed CD.

- Changed the persistence property of the description attribute to @Lob
- Added Product attribute with @ManyToOne@JoinColumn(name='product\_id') relation
- **MailAddress.java**
  - New class with attributes String address and Person owner with a @ManyToOne relation
- **Person.java**
  - Removed attribute email
  - Added attribute Set<MailAddress> mailAddresses with a @JoinTable(name = 'mailaddress', joinColumns = @JoinColumn(name = 'id'), inverseJoinColumns = @JoinColumn(name = 'owner\_id')) relation definition
  - Added attributes String cvsUserName and String mailAddress as @Transient

## Communication Path Transformation

Problem reports consist of a set of properties such as the classification, the priority, or the date when the report has been opened. In order to discuss an issue, comments can be added to the problem report until it gets closed. We assume these discussions as communication between the commentators. The problem report object model does not consider any communication objects and their corresponding paths. The raw problem report data is transformed to match the communication model as illustrated in Figure 4.4. The approach of the algorithm is specified in Section 3.4.2 and the transformation process described below.

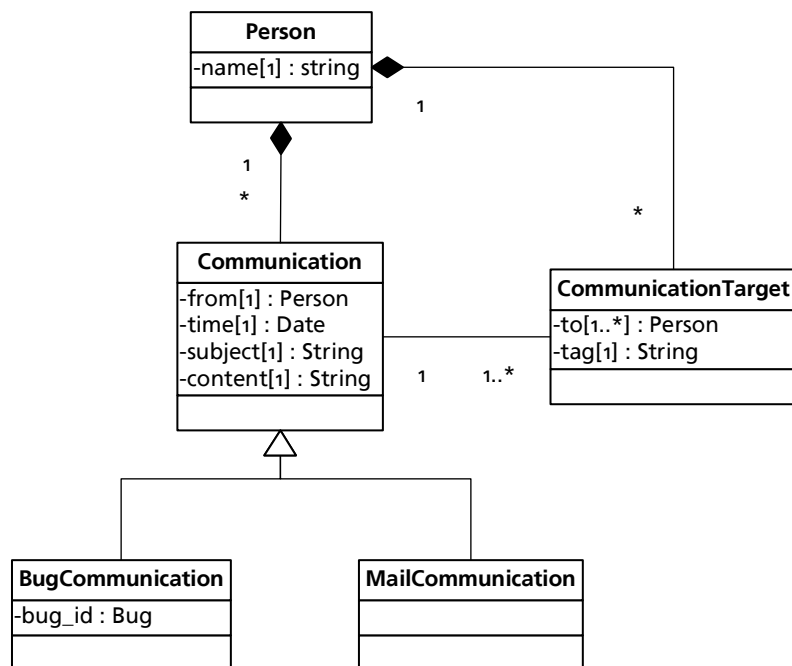


Figure 4.4: Communication Model

The data schema *bugs* contains the raw data of the imported problem reports. The following Listing 4.3 lists the process steps to derive the communication paths from the problem reports.

```

1 CREATE TABLE bugs.comm LIKE mails.comm;
2 CREATE TABLE bugs.comm_target LIKE mails.comm_target;
3 CREATE TABLE bugs.logging LIKE mails.logging;
4 SELECT bugs.extract_bugs_and_ccs();
5 SELECT bugs.extract_comment_comm();
6 CREATE TABLE bugs.comm_target_tmp LIKE bugs.comm_target;
7 INSERT INTO mails.comm_target_tmp
8     SELECT datasetid, comm_id, comm_to, tag
9     FROM mails.comm_target_bkp
10    GROUP BY datasetid, comm_id, comm_to, tag;
11 DROP TABLE bugs.comm_target;
12 RENAME TABLE bugs.comm_target_tmp TO bugs.comm_target;

```

**Listing 4.3:** Processing Steps of the Communication Path Transformation

## 4.3 Data Integration

The person is the subject connecting the three models, as it is illustrated in the interaction target model (see Figure 3.3). The underlying data extractions have different approaches regarding the identification and characterization of a person. The personal information appearing within CVS logs, problem reports or mails, are the name of the person, the mail address and the CVS user name. In order to avoid falsification of the social network representations, the integrated data object model ideally holds only one object per person in the database. The person object model is defined as consisting of at least a unique name, null or several mail addresses and null or several CVS user names. The details are outlined in Section 4.3.1.

The mail address contains an additional information about the person. In case of a business mail address, the prefix of the address determines the company somebody works for. For that the domain details can be extracted, they may not be hidden.

### Challenges

In the virtual world of the Internet it is easy to create an email account, or even several ones. To identify a person working with different email addresses requires to know the person's real name. Imagine the email address *tevion48x@yahoo.com* appearing in the Cc: list of a problem report and the owner of this address reports a problem as *Agent Smith* using the email address *haefeuser@yahoo.com*. It is not trivial to identify this person as *John Smith*. To make another example for a more active user, *Chris McGee* uses *cbmcgee@ca.ibm.com*, *jeffl@informaldata.com* and *sirnewton\_01@yahoo.ca* as his email addresses.

Regarding the grouping of teams using the domain segment, we face another problem. The mailing list archive MHonArc provides a spam mode configuration that deters spam address harvesters by hiding the address domains. This results in displaying an email addresses like *www.jsports.org@xxxxxxxxxx*.

### 4.3.1 Person Allocation

Since the circle of Bugzilla editors is the biggest of the three person sub-groups, this is taken as base for the mapping. Besides the bug parser and importer functionality, `org.seal.snanalyzer` contains an extended implementation of a person finder and an allocation algorithm. Its aim is to set the correct relations to a problem report and to have every person stored only once within the database.

#### Person Object Model

The algorithm is processed every time a person attribute is set, for example for an assignee of a problem report or a commentator. It checks each person by analyzing the name and email address and comparing them to the existing person objects contained in the object model that will be stored later.

First the email address is looked up in the object model. If it is found, this person already exist and no further actions are required. In every other case, the mail address is analyzed. The real name of the person is tried to be extracted from the prefix of the email address, because a person's name is often contained therein. In some cases the real name can not be derived from the prefix, because it contains *bugzilla* or *eclipse* or consists of only an alias or a nick name as *fanyuz@cn.ibm.com*. It is assumed that a name consists of at least two words and that they are separated by a dot or underscore within an email address prefix. Then the implemented helper class tries to find this persons's real name by searching the Bugzilla database for an entry reported using this email address. If the query succeeds, the according bug page in HTML format is parsed, in order to read the name that is indicated previous to the email address. If a person object with this name is found in the object model, the new email address is added to the list of email addresses of this person. In every other case the person is assumed to be unknown and a new object is created. The two main methods are listed in Listing 4.4 and 4.5.

```
/**
 * Organizes the person's roles:
 * Reporter, Assignee, CCs, Commentators.
 * For every person, the getAssoziation(Person p) is called,
 * to set the corrent relation.
 */
private Person getAssoziation(Person toCheck) {
    String mailAddress = toCheck.getMailAddress();
    Person returnPerson = null;

    MailAddress mail = mEmails.get(mailAddress);
    if (mail != null) { // existing Mail --> known Person
        returnPerson = mail.getOwner();
    }
    else { // new Mail
        String prefix = getMailPrefix(mailAddress);
        String realName = Person.convertToFLUCase(prefix);
        Person found = mPersons.get(realName);

        if (found != null) { // existing Person - new Mail
```

```

        MailAddress newMail = new MailAddress();
        newMail.setAddress(mailAddress);
        newMail.setOwner(found);
        mEmails.put(newMail.getAddress(), newMail);
        returnPerson = found;
    }
    else { // new Person - new Mail
        // try to get real name via html bug page
        if (prefix.contains("bugzilla") || prefix.contains("eclipse") || !
            prefix.contains(" ")) {
            try {
                PersonFinder finder = new PersonFinder();
                finder.search(mailAddress);
                realName = Person.convertToFLUCase(finder.getRealName());
            } catch (Exception e) {
                e.printStackTrace();
            }
            finally {
                if (realName.length() == 0) {
                    realName = mailAddress;
                }
            }
        }
        found = mPersons.get(realName);
        if (found == null) { // still new person
            found = new Person();
            found.setName(realName);
            mPersons.put(found.getName(), found);
        }

        MailAddress newMail = new MailAddress();
        newMail.setAddress(mailAddress);
        newMail.setOwner(found);
        mEmails.put(newMail.getAddress(), newMail);
        returnPerson = found;
    }
}
return returnPerson;
}

```

**Listing 4.4:** Implementation of the getAssoziation Method within BugModelBuilder.java

```

/**
 * Derives the name of the mail address's owner
 * by removing special characters.
 * @param email - email address in String format
 * @return the modified prefix (e.g. john arthone)
 */
private static String getMailPrefix(String email) {
    String name = email.substring(0, email.indexOf("@"));
    name = name.replace(".", " ");
    name = name.replace("_", " ");
    name = name.toLowerCase();
    name = name.trim();
    return name;
}

```

**Listing 4.5:** Implementation of the getMailPrefix Method within BugModelBuilder.java

## CVS User Mapping

The Evolizer's CVS importer stores the person information in an object called author. The name indicates the CVS user name, but not the real name of the person. In order to connect the persons that add and edit problem reports to the ones contributing to the code development, the author object is extended by a person\_id. Since CVS user names do not contain special characters to indicate separate words, the mapping has to be done manually. Due to the relative small amount of CVS users, implementing a similarity algorithm would have been too expensive. Simply comparing the author's CVS user name with the names in the Bugzilla person source helps to find the according entry. In case no adequate name is found the new person has to be manually added to the database.

## Mailing List Users

As illustrated in the iQuest schema in Appendix 4.3, the iQuets data model does not consider any key that is independent of the person's name or email address. The relation between the persons and to the communication objects is done using these strings. To map the mail communication to the person model, the *comm\_from* and *comm\_to* have to be replaced by the id of the according person's id, contained in the person source that is previously built during the Bugzilla import, and extended while extracting mailing lists. The simplest way is to process the bug communication within the iQuest tool and to create a *Communication View* combining the mail and bug communication. Then the functionality of the mail address redefinition allows to simply map for example 'john\_arthorne@xxxxxxxxxx' to 'john arthone'. The algorithm compares the name with the prefix of a mail address and makes proposals based on similarity measures. The validation is made manually. In a following step this mapping (stored in a table called *name\_change*) is joined with the existing person model. The person's id is set to the *mail* object attribute *comm\_from* and the *mail\_target* object attribute *comm\_to* respectively. If there are mailing list users left, for whom there exist no accurate person, a new object has to be created and mapped manually.

### 4.3.2 Delimitation & Timeframe Alignment

The integrated database registers 2'615 persons. In order to evaluate the visualization, the groups of people have to be limited. The development team that consists of 27 people, is selected to be surveyed. The following 4.1 lists the available data and outlines the delimitation impacts. The modification reports are not affected while there is an extensive reduction of data regarding the community members and the communication objects.

In order to compare equivalent data sets, the available timeframes have to be aligned. The period over which all data sets contain data, lasts from 2002-01-28 to 2006-11-13. The processing of cleaning and aligning the data can be found in B.3.

	report_all	report_core
Person Related Data		
person	2'615	27
mailaddress	2'681	28
author	94	93
Collaboration Data		
plugin	17	17
source_unit	997	997
revisions	7'479	7'479
component	3	3
Communication Data		
bug_comm	28'131	7'907
bug_target	58'927	22'429
mail_comm	891	102
mail_target	786	117

**Table 4.1:** Project Data Consolidation

Table 4.1 does not yet show the total number of interactions. If the sender does equal the receiver, which can be a result of dissolving the mail thread or when somebody writes many comments within the same problem report, this is not considered as interaction. The total number of interactions is listed the the below.

**Mails Communication:** 101  
**Bugzilla Communication:** 11'081  
**Modification Reports:** 7'479

## 4.4 Summary

This chapter describes the required steps to integrate mailing lists, Bugzilla reports and CVS logs into a integrated database. Section 4.2 outlines the processed actions and the data meta model in detail for each data source. To extract CVS logs, Section 4.2.1 presents the Evolizer CVS Importer and discusses its restrictions. Possible improvements and the implemented extensions to obtain the data as desired are described in Appendix B.3 and Section 4.2.1. Mailing lists have been extracted using iQuest. 4.2.2 lists the handling of the tool and the extensions. To gain the Bugzilla reports, an importer algorithm was implemented which is described in Section 4.2.3. This section includes the communication transformation, since they are not modeled explicitly within the

problem report data meta model. Section 4.3 outlines the final integration of all extracted data sources. We discuss the challenges and provide algorithms to overcome them the best feasible.

The following Chapter 5 presents the implemented visualization by first discussing different visualization styles. The subsequent sections outline the dimensions of the underlying data and describes the concepts of the visualization elements. This includes the design of the graphs that will be displayed by the tool. Finally, the visualization tool is explained by making examples. The implementation of the Cockpit will be evaluated in Chapter 6.3.



# Visualization

Analyzing large data sets demands for coherent and comprehensible representations. The human brain is more capable of processing visual information than interpreting data in its raw format or therefrom derived figures like statistical analysis. In order to draw conclusions and examine complex or abstract data sets, the selection of an adequate visualization instrument is crucial, especially when dealing with time-dependent data.

This chapter first outlines different approaches of information visualization with focus on representing interaction networks. Based on the findings, the motivation for the implementation of the Social Network Analysis Cockpit are stated. The subsequent section addresses the dimension and abstraction issues to find an approach of visualizing interaction networks without disregarding the dynamics. The next section then presents the concepts underlying the visualization. The aim of the implementation is to help people make sense of the project and its environment. Therefore, the last section finally explains the implemented information coding and how to interpret the visualization.

## 5.1 Visualization Styles

The range of visualization types and alternatives is wide. The optimal style depends on the complexity of the underlying data set and on the intended objective. Furthermore, large data sets can not be represented completely and in every detail. Visualizing data demands for a certain level of abstraction. In order to approach the issue of representing interaction networks in an understandable way, this section analyses the practices and alternatives. All of the three following evaluations represent data that has been extracted within the processing described in Chapter 4.

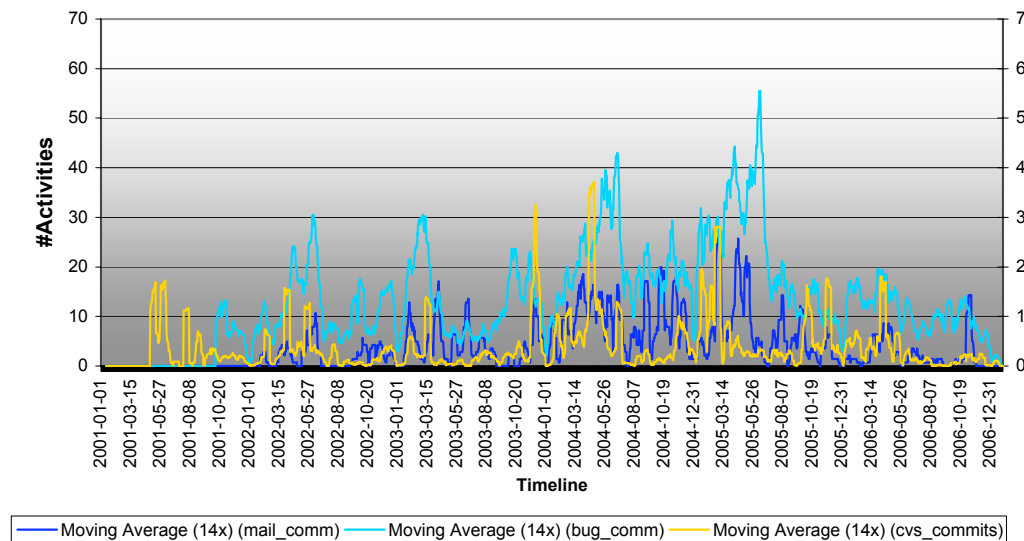
### 5.1.1 Visualizing Time-Dependency

A significant requirement regarding dynamic data sets is the visualization of the evolution, such as that the observer is able to identify the transformations. One possibility to visualize time-dependent data is to span the time over one dimension. While assuming a two dimensional space, only the second dimension is left to represent the data. This is a very limiting alternative as the following sample sustains.

The first sample, Figure 5.1, illustrates the distribution of the interactions occurring within a project. It illustrates a diagram that is based on the time series of the occurrence of mails, problem and modification reports.<sup>1</sup> All three lines show the moving average of the distribution. The

---

<sup>1</sup>Details regarding the data and project selection can be found in Chapter 6.3.

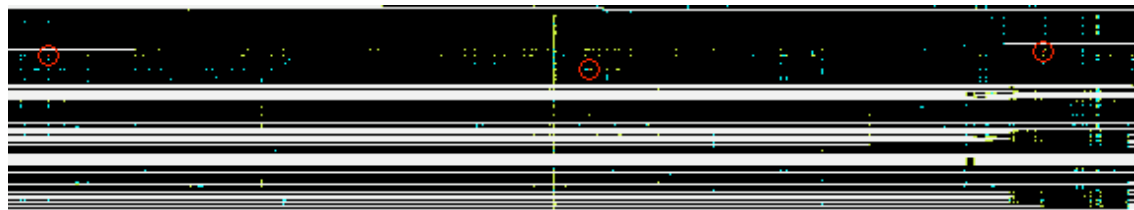


**Figure 5.1:** Distribution of Collaboration Actions

yellow line represents the distribution of the modification reports for the selected data set. The light blue line represents the communication activity arising from problem report discussions and the dark blue line indicates the mail communication. The distribution of the mail communication is scaled by factor ten compared to the other two distributions. It seems as this does confirm the initial hypothesis, that the communication about development tasks takes places within problem report tools and therefore is more frequent than mail communication. Furthermore, the curves do not correlate, neither do they indicate any significance. Consequently, this representation does not fulfill the requirements of visualizing dynamic interaction networks.

### 5.1.2 Visualizing Collaboration Patterns

In order to use the second dimension to illustrate the states of a project, the data source requires to be stable. For software development, this could be files that are lined up spanning the time dimension. The following sample illustrates this alternative. In addition to the first sample, by coloring the different states, further information can be coded.



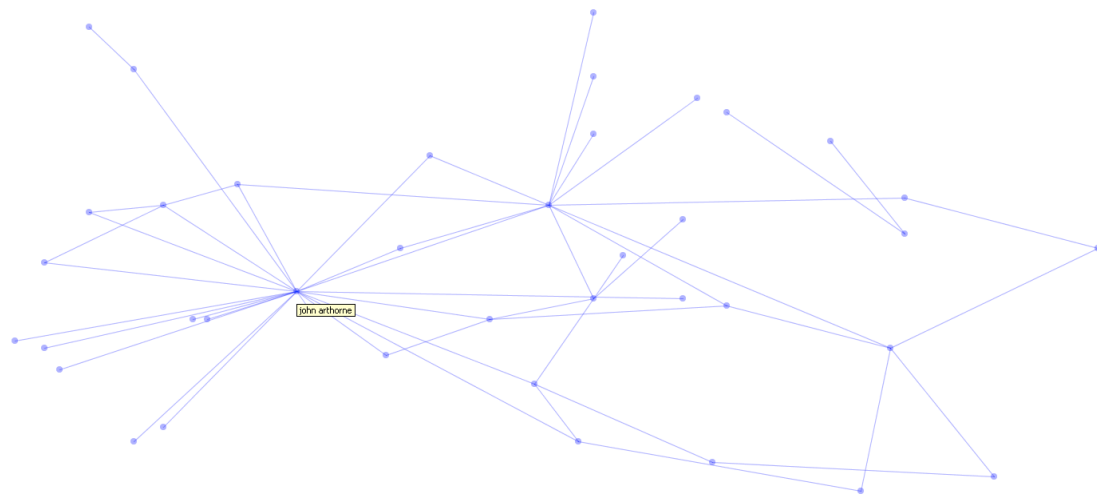
**Figure 5.2:** Source Code File Modification over Time

Figure 5.2 tries to show patterns regarding the source code modifications. It represents a sample of the overview graphics of all analyzed Eclipse Plug-ins and was generated by our prototype

visualization tool.<sup>2</sup> The different roles of the developer are colored in order to easier find behavioral patterns. The horizontal axis indicates the timeline (increasing from left to right). Each file represent a horizontal lines. The files are grouped by projects.<sup>3</sup> Within the project, the files are ordered according to the organization inside the project's file tree. The files that are contained in the same source package are displayed close to each other.<sup>4</sup> A colored pixel denotes an action or the status of a file at the corresponding point in time. A black pixel indicates a file that exists, but which is not affected by any action. A blank pixel implies the file being inexistent. The modification report actions are represented by the colors red, green and cyan. A green action is an ordinary source code modification, done by the owner of the file. A cyan actions indicate a code modifications performed by somebody not being the owner. Based on this condition, a red colored pixel (highlighted by a circle) points at a change of ownership. In order to perceive the project, the use of the second dimension to illustrate each file's evolution and the developer's role helps. But the collaboration information or further qualities of the project are still missing. The overview<sup>2</sup> finally assists to find interesting spots that show a higher activity. However, the significance is still inadequate to represent interaction network information.

### 5.1.3 Visualizing Social Networks

Social networks are represented by graphs. Compared to the previous two samples, the illustration of static graphs does not support to take the time dimension into consideration. Otherwise, the scope of potential information being coded within a representation is much bigger. However, the evolution can not be depicted with static graph representations.



**Figure 5.3:** iQuest Social Network Graph

Figure 5.3 shows the conventional representation of a social network. The sample illustrates the social network based on only the mail communication. For this sampling purpose, the ex-

<sup>2</sup>The complete graphics can be found in Appendix ?? . The source code of the java implementation can be found on the enclosed CD.

<sup>3</sup>Project in this context means a working project configured within the Eclipse workspace.

<sup>4</sup>Note that modifications within the project's directory structure are not considered. The organization represents the standing at the time of the data extraction.

tracted data set was processed by the iQuest analyzer tool.<sup>5</sup> The nodes represent community members and the edges indicate mail communication. Although the figure allows a person to be identified within the community, the graph does not contain any further information (for example about the development contribution). Since the focus of this thesis is to combine the social network with the project's contribution configuration, this representation is not sufficient for further analysis.

### 5.1.4 Motivation

The analyzed tools and approaches do not fulfill the demanded requirements without losing too much information or having no alternatives than to process the data as designed. For example, the social network can be processed with the iQuest tool, but not without pre-processing the data. The purpose of iQuest does not completely correspond to the schema of this thesis. To conclude, the implementation of our Social Network Analysis Cockpit intends to enable the analysis of the extracted data set.

## 5.2 Dimensions & Abstraction

Visualizing large or complex data sets and including all its aspects is not feasible, but requires a certain level of abstraction. The gained collaboration data set spans the time and constellation dimension. In order to represent the data in an understandable way, the difficulty is to identify the optimal level of abstraction. This section briefly outlines the significance regarding the extracted data set.

### 5.2.1 Time

In order to visualize dynamics, the underlying data requires to contain date specific information. Each time-dependent data object needs to have a date attribute. Regarding the interaction model, this time-dependency affects the mail communication and the contribution to problem- and modification reports - thus, all interaction objects that have been introduced in Section 3.4. This implies the communication and revision date to be memorized, since these interactions all take place at a particular time.

### 5.2.2 Configuration

The configuration is determined by the state of the interaction network at a given point in time. This includes the communication paths that represent the interactions between the project members and the modification actions that relate a source code file to the performing developer. The resulting interactions are defined by their type indicating mail communication or problem- and modification report actions. Within the scope of the visualization, the content of the communication is not considered. Therefore, mails with the same subject and of the same thread are not illustrated as such. The same applies to problem reports and their comments. Regarding modification reports, the abstraction is set to the project level because the Eclipse Plug-ins are organized in a modular way. The modularization is listed in Appendix D. and also applies to other software

---

<sup>5</sup>The formatted and processable data is contained on the enclosed CD.

development projects. Details about the Plug-ins can be found in Chapter 6.3. To preserve additional characteristics that represent the project's situation, like revision specific features as the alien commit, are added to the social network.<sup>6</sup>

## 5.3 Concepts

The aim of this section is to present an approach to combine the dimensions outlined before and to find a suitable representation form for interaction networks. This interaction network graph contains both the social (communication) and the collaboration structure of the network.

### 5.3.1 Social Network Graph

A social network graph is defined by a number of interconnected actors. Each actor is represented by a node within the graph. The connections between the actors are based on their communication within mail traffic or problem reports. The approach of the transformation into communication paths is outlined in Section 3.4.2. Therefore, whenever two actors communicate, they get related to each other. These relations determine the edges of the communication graph.

#### Graph Quantities

A graph consists of nodes and edges that determine its structure. In order to visualize a social network graph, characteristics of the graph need to be measured to define the shape for the graphical representation. This includes the length of a path between two actors or the position of a node on the drawing. Finally, the arrangement of the graph elements is depending on the dimension of the illustration. The following measures quantify the nodes and edges and determine how a social network graph looks like.

- *Betweenness Centrality*: Determines the extent of direct connections to actors that are not connected to each other. An actor occurring on many shortest paths between the other actors has a higher betweenness compared to others. To use again the notation of Gladwell an actor with a high betweenness (compared to the others) is more likely to be a *Connector*.<sup>7</sup> This states the strategic importance of an actor related to the information paths.
- *Degree Centrality*: Defines the number of ties between an actor and its direct neighbors. Since the indirect connected actors are not considered, this does not declare an actor being more or less important. The algorithms to compute the degree centrality differ in terms of consideration of the edges. We weight incoming and outgoing edges the same.
- *Closeness Centrality*: States how close an actor is to the others in the network, by measuring the length of all shortest paths from this actor to all others. This measure indicates where information is processed through.
- *Path Length*: Determines the distance between two connected actors and refers to the relative average path length.

The nodes and edges are arranged on the drawing according to the calculated features. With an increasing number of nodes and edges in the graph, the complexity arises likewise. The more complex the graph gets, the longer the calculation algorithm that computes the position of each

<sup>6</sup>See Section 3.4.1 for the definition and Section 4.2.1 for the implementation.

<sup>7</sup>His book *The Tipping Point* is mentioned in Chapter 3.1.

node based on its quality measures lasts. Furthermore, the particular quality sets compete because every node is qualified regarding different aspects and the length of edges is defined as well.

### 5.3.2 Collaboration Network Graph

The collaboration network graph is built using the same methods applied to social network graphs. Additional to the actors of the social network, the collaboration network contains nodes that represent work packages. The edges of a collaboration network graph are determined by the contribution of a developer to a source code file that is contained in that specific work package. The second reason for an edge relating an actor node with a work package node, is the illustration of source code ownership. If an actor of the network owns a part of a work package that is being modified, this is represented by a connection in between.

### 5.3.3 Dynamics - Time Window Selection

An interaction network graph always structures a particular state. Since the communication between people and the collaboration contribution are distributed over time, the graph ideally contains network information covering more than two hours. Furthermore, the time window size can differ between the communication and the collaboration graph. The challenge is to find an appropriate size of the time window that supports the intended representation without blurring the significance. Besides the window size, the shifting position needs to be defined. The possible time shiftings range from the smallest time unit to the total size of the window.

#### Animation

In order to show the evolution of the networks, an animated visualization enables the project environment to get explored.

## 5.4 Social Network Analysis Cockpit

The aim of this section is to describe the implementation of the visualization tool, called the Social Network Analysis Cockpit (SNA Cockpit). The tool is implemented as Eclipse Plug-in and launched by selecting the *Dynamic View* of the *Social Network Analysis* view.<sup>8</sup> The graph visualization is implemented using yFiles [yG].<sup>9</sup> The SNA Cockpit consists of four areas that are described in the following sub sections.

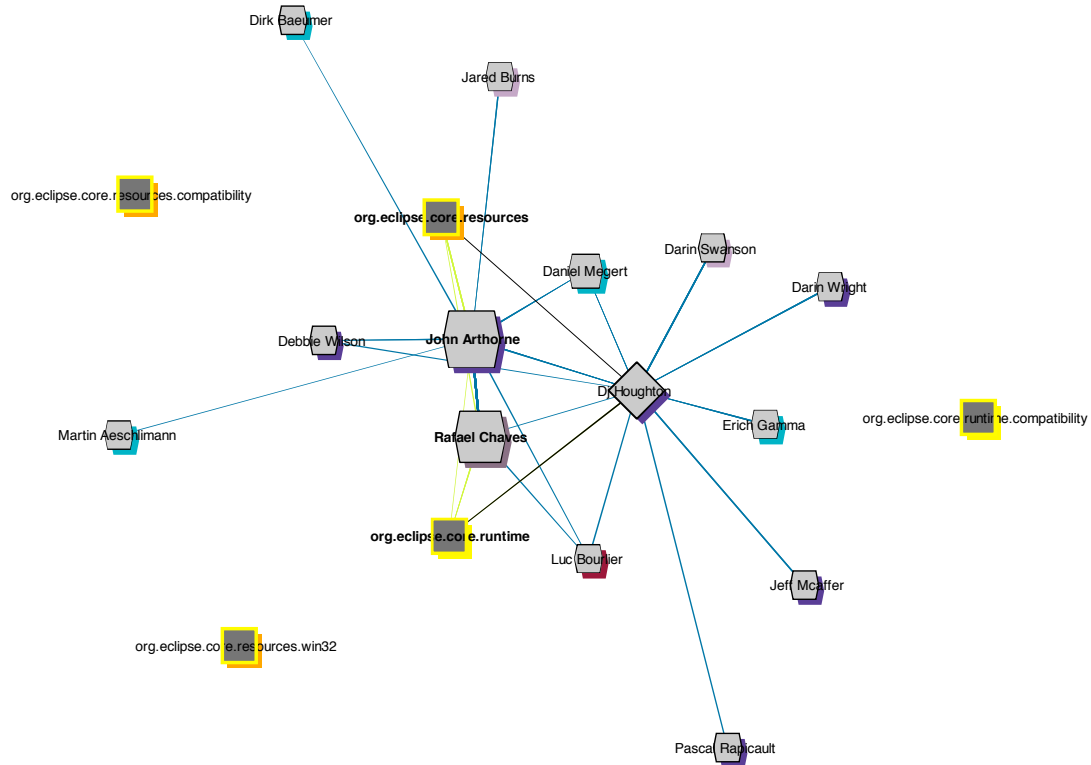
### 5.4.1 Graphs

The main component of the SNA Cockpit is the graph panel containing the network representation. The visualization of the interaction graph is based on the methods outlined previously in this chapter. The values of the calculations described in Section 5.3.1 are not illustrated as the graph implicates these qualities within its representation.

This sub-section describes the different elements of the interaction network graph, by making examples and explaining the meanings. It examines the nodes representing actors that play different roles and the work packages. Furthermore, the different type of actions and the depiction of

<sup>8</sup>See Appendix C.1.4 for the description of how to start the tool.

<sup>9</sup>Version: yFiles Complete for Java 2.5.



**Figure 5.4:** Social Network Graph

project specific status information is outlined. On the left a little extract of the explanation graph is pictured, annotated by the text on the right.

The following list outlines the meaning of the nodes and edges contained in the graph. Furthermore, for each type of node or edge, the applying rules are explained.

**Nodes** represent objects that are valid within the defined time frame. The design of the node implies the characteristics of the represented object.

- **Communicating Person:** A person who is communicating via mail or active within problem reporting. Mail communication encloses two different roles: Composer or receiver of the mail. Within problem reporting, a person can play the following roles that are considered: Reporter, assignee, 'CC' or commentator.
- **Working Person:** A person who is committing source code modifications.
- **Work Package:** A work package represents an Eclipse workbench project. A work package is displayed, when either some of its source files are modified or a Bugzilla report to a component this work package belongs to, is opened.

**Edges** either represent an interaction, an action or a property that occurs or is valid within the defined time frame.

- **Mail Communication** connects the sender to the receiver. The embodiment of somebody being both the sender and the receiver of a mail is not represented. This applies to all types of communication.

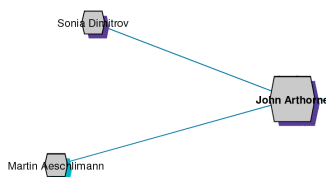


- **Opening a Bugzilla Report** links the reporter to the assignee of the issue. Furthermore, every person recorded as 'CC' is connected to the reporting person.
- **Commenting a Bugzilla Reports** connects the commentator to the assignee, to the reporter and to all preceding commentators.
- **Committing Source Code Modifications** links the developer to the work package the modified file belongs to.
- **Owning Source Code** connects the owning person of a source code file that is being modified, to the work package the file belongs to.

## Actors

An actor can play different roles within the project. A project member is illustrated by a grey actor node and labelled with the name. The default shape is a hexagon and the border color is always black. Whether an author is involved in communication or not is not implied by the node representation.

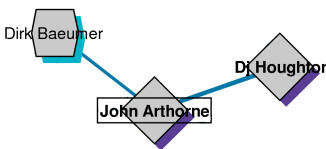
The size of an actor depends on the degree centrality that is determined by the number of incoming and outgoing edges. The bigger the node is, the more this actor communicates. The size only depends on the communication frequency.



The shadow of a node illustrates the sub-group to which the actor belongs. Sub-groups are formed by extracting the domain string from the email address. The color of the shadow is defined by a derivation of the hash code of the domain string. Therefore, the shadows of all actors of the same domain are colored uniformly.



Regarding the code contribution, a developer can play two different roles. Either he is the owner of the modified source file or he performs an alien commit, that is defined as a source code modification, performed by somebody not being the owner. The owner of a file is illustrated by shaping the node as diamond and drawing a black line relating the owner to the work package the file belongs to. These *owner* edges are the only ones not representing an interaction.



If the contribution of a developer within a modification report results in an change of owner, this is illustrated by framing the developer's node label. With respect to the path length between actors, the extent of exchanged messages defines the length of the edge. The more frequent two actors communicate, the closer they

are arranged. John and DJ communicated more often with each other than John and Dirk.

An alien commit is illustrated by setting the font style of the involved nodes to bold. This applies to the developer node and the work package likewise.





## Work Packages

**org.eclipse.core.runtime**

A work package is illustrated by a dark grey rectangle. The

default line color is yellow and the default color of the shadow is light grey. A bold labelled work package indicates that a file has been modified by somebody not being its owner.

The border color represents a risk measurement regarding the modification reports. Whenever a commit message of a modification report contains a bug number, the fix is asserted to implying a lower risk of resulting in a defect compared to a revision without containing a bug number, which is seen as a modification with no declared intension. The more red the line is, the higher is the number of revisions not related to a problem report.

The shadow indicates the extent of reported problem reports to the component this work package belongs to. Only the problem reports whose resolution type is classified with either FIXED, REMIND, WON'TFIX or WORKSFORME are considered.<sup>10</sup> The more red the shadow gets, the more bugs are reported. The Bugzilla-project mapping can be found in Appendix D. Whenever a work package is depicted but not connected to any developer, this indicates that the work package is being used.

**org.eclipse.core.resources**

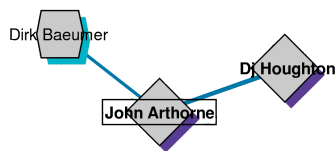
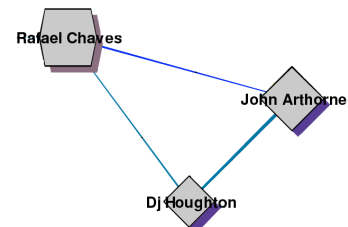
## Actions



A software development contribution action is indicated by a green line and represents the extent of modification reports performed by a developer. The edge is put between the developer and the work package the modified file belongs to. The thickness of the line indicates the number of modification reports. The

thicker the line is, the more modifications are performed. In case of an alien commit, the node label font style of the developer is changed to bold.

All other edges indicate communication. The communication arising from problem report discussions are colored steel blue while mail communication is colored royal blue. The thickness depends on the extent of communication between the relating actors.



A change of ownership is indicated by framing the developer and new owner's node label. The sample shows John Arthorne as the new owner of a file he has modified. Since a change of owner implies an alien commit, the text style of the developer's node label is changed to bold as well.

<sup>10</sup>It does not seem to be obvious to take the resolution type instead of the severity classification. The reason for this is related to the handling of the problem report features and it is outlined in Appendix C.1.1.

## The Analysis Panel

Active Timeframe:	15.12.2004 - 22.12.2004
Surrounding Timeframe:	08.12.2004 - 29.12.2004
# of Bug Communications:	28
# of Mail Communications:	0
# of Revisions:	28
# of Alien Commits:	12 [0.43]
# whereof Owner Changes:	1 [0.04]
org.eclipse.core.expressions	2 [1.0]
org.eclipse.core.resources	18 [0.78]
org.eclipse.core.runtime	3 [1.0]
# Bugs Timeframe:	15.12.2004 - 29.12.2004
# Bugs Reported:	16
Reported Bugs @ Resources	8
Reported Bugs @ Runtime	8
	gmail.com
	ca.ibm.com
	us.ibm.com
	ch.ibm.com

**Figure 5.5:** SNA Cockpit Analysis Panel

In order to learn something about the project, the collaboration network is extended by problem report classifications. A high number of reported problems for example indicate an active usage of the component. Valuable features of problem reports that state the project's situation, are either incorporated into the representation or are displayed as figures on the Analysis Panel. It reports a lot of project status details referring to the actual representation of the interaction network within the SNA Cockpit. The panel is part of the cockpit and illustrated in Figure 5.5. The panel consists of seven section whose objectives are described in the following list.

**time frames:** Displays the active and the surrounding time frame that is set on the control panel. The active time frame indicates the time window size for the collaboration network, whereas the surrounding time frame defines the time window size for the communication network. Depending on the user selection, these two time frames can completely overlap.

**Number of Interactions:** Indicates the total number of occurring actions within the defined time

frame for each interaction type.

**Contribution Types:** Shows the total number of alien commits that occur within the collaboration network during the active time frame and the therein contained number of changes of ownership. The figure in brackets indicates the ratio to the total number of revisions.

**Risk Degree:** Lists the work packages that contain a source code file that has been modified without indicating a bug number in the commit message. The number in parentheses indicate the ratio (in proportion to the total number of modification reports within the corresponding work package).

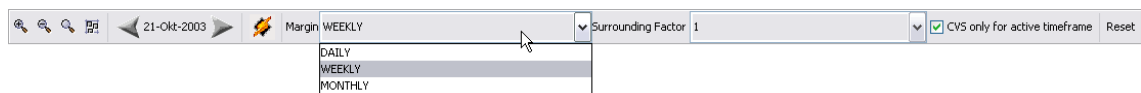
**Problem Reporting:** Indicates the time frame over which the reported problems are taken into account. The start of the time frame equals the start of the modification reporting time frame, while the end corresponds to the end of the communication time frame. Besides the period, the total number of reported problems that are considered are indicated.

**Bugzilla Components:** The reported problems are grouped by components that are listed in case of occurring reports. The number indicates the quantity.

**Domains:** Based on the actor's email address, the domain is derived and defines the color of the shadow of an actor's node. The listing shows the name of the domain and the according color.

## 5.4.2 The User Interface

In order to enable an accessible handling of the visualization, the graphical user interface (GUI) provides features to define the represented data. As outlined in Section 5.3.3 the selection of the time window is crucial to get a feasible representation that enables analysis. Due to the dynamics of the project and its evolution, the optimal window size varies. Furthermore, the window size of the communication and the collaboration network are desired to be overlapping. Development tasks relating communication are assumed to happen also before the modification is done to the source file. In order to cope with this dynamics, the control panel features the selection of the window size supposed to be displayed. Figure 5.6 shows the control panel and the following list describes the provided features.



**Figure 5.6:** SNA Cockpit Control

**Graph Window Zoom:** The first two buttons allow the user to zoom in or out the graph panel. To zoom a chosen area of the graph window, the third button enables to select the desired area with the mouse pointer. The fourth button is intended to reset the graph on window size.

**Moving Forward and Backward:** The arrow buttons enable to move along the time dimension, based on the selected window size. By using the forward option the required data is fetched and the graph is recalculated.

**Calendar:** With the pop-up calendar the desired date can be selected. By scrolling through the time window slices, the calendar always displays the start time of the active time frame.

**Rearrange:** The flash button rearranges the graph. Since the calculation algorithm to compute the optimal layout of the graph takes some time, the flash button enables to recall it.

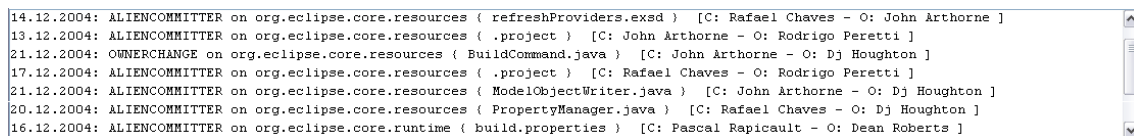
**Margin:** The margin drop down menu enables the selection of the active time frame for the collaboration data, whose time window size can either set to daily, weekly or monthly. The default value is weekly.

**Surrounding Factor:** The surrounding factor determines the factor of which the active time frame is extended. This is the time frame that is decisive for the communication network. To make an example, consider the second week of this year as selected as active time frame. A surrounding factor of one will extend the active time frame by the week prior and posterior. The default value is one.

**CVS Flag:** This flag defines if the active time frame is adjusted to the overall time frame. If checked, the collaboration network span only the active time frame. If inactivated, the collaboration network span the same time frame as the communication network.

**Reset:** The reset button enables to restore the graph to the initial selected date.

## The Modification Reports' Detail Display



```

14.12.2004: ALIENCOMMITTER on org.eclipse.core.resources ( refreshProviders.exsd ) [C: Rafael Chaves - O: John Arthorne ]
13.12.2004: ALIENCOMMITTER on org.eclipse.core.resources ( .project ) [C: John Arthorne - O: Rodrigo Peretti ]
21.12.2004: OWNERCHANGE on org.eclipse.core.resources ( BuildCommand.java ) [C: John Arthorne - O: Dj Houghton ]
17.12.2004: ALIENCOMMITTER on org.eclipse.core.resources ( .project ) [C: Rafael Chaves - O: Rodrigo Peretti ]
21.12.2004: ALIENCOMMITTER on org.eclipse.core.resources ( ModelObjectWriter.java ) [C: John Arthorne - O: Dj Houghton ]
20.12.2004: ALIENCOMMITTER on org.eclipse.core.resources ( PropertyManager.java ) [C: Rafael Chaves - O: Dj Houghton ]
16.12.2004: ALIENCOMMITTER on org.eclipse.core.runtime ( build.properties ) [C: Pascal Rapicault - O: Dean Roberts ]
  
```

**Figure 5.7:** The Modification Reports' Detail Display

Since the representation of the modification reports are abstracted to the work package level, details that are not included in the graph are not visible. In order to display additional details about the modification, the cockpit contains a scrollable pane. Figure 5.7 shows the panel that lists in case of an alien commit or a change of ownership, the details of the transaction. Each lists the date of the revision, the type of the modification, the affected plugin and class, and the names of the modifier and the owner of the file.

## 5.5 Summary

This chapter outlines the implemented visualization. First, we have analyzed and discussed possible visualization styles in order to select an appropriate approach to implement the evolving networks. The second section addresses the dimensions and abstraction issue. The subsequent section outlines the dimensions of the underlying data and describes the concepts of the visualization elements. This includes the design of the graphs that will be displayed by the tool. Finally, the Social Network Analysis Cockpit is explained by making examples.

The implementation of the Cockpit will be evaluated on the data of Eclipse in the following Chapter 6.3.

# Case Study & Evaluation

This chapter analyzes the Eclipse Platform Core community and evaluates the Social Network Analysis Cockpit. The first section outlines the requirements and the assessment regarding the project selection. Next, we present the case study by analyzing features of the different data sources and we give an overview of the community. This section aims at gaining insights into its structure of the community and a better understanding of the project.

The evaluation of the Social Network Analysis Cockpit is performed with the core development team. The implementation is tested by addressing the questions introduced in Section 3.1. The objective of the evaluation is to find answers to the given questions. First, we introduce and explain several patterns. Within the following sections the perspectives described in 3.1 are addressed and interpreted.

## 6.1 Project Selection

This section outlines the requirements of the project selection with the objective to point at the crucial factors and to give hints for further extensions of the data set.

### 6.1.1 Requirements

Applying the approach described in Chapter 3 requires access to the data, and the fulfilling of the conditions outlined in this section. Considering the availability of all three data sources with respect to a similar time period, avoids the risk of clashing data. A source being moved to another storage or the reorganization of a system and its processes can cause missing or misleading reports. A stable and complete historic and current data set is crucial for the subsequent processing. Besides the availability of the data sources and their history over the same period, the data needs to have all required features available, in order to derive the model previously introduced in Chapter 3. The data extraction and integration steps are covered by Chapter 4.

Characteristics qualifying the project, extend these technical requirements. Potential projects are tested for their number of developers, their present lifetime and the amount of existing source code. Available developer lists with information about the code contributors and details about the community would be helpful, but either do not exist, are not available or are restricted due to legal regulations.

### 6.1.2 Project Assessment

There exist a vast number of open source software projects whose data is accessible via the Internet. FLOSSmole [FLO04] is a platform that provides high-level summary reports and raw data about open source projects, mainly from SourceForge.net [OST04]. The aim is to establish a collaborative research data base, providing free/libre/open source project data collections for analysis purpose. Many of the resultant research papers discuss projects on a higher level than within this thesis. Besides SourceForge.net, there exist more large open source platforms such as mozilla.org and eclipse.org. Projects hosted on such platforms are based on the open source applications running on these systems. The mailing lists at SourceForge.net are managed by GNU Mailman, whose online data representation is user friendly with a lot of features, but complicated to parse. At mozilla.org newsgroups are used, but there exist no mailing lists for development teams. Finally the Eclipse Project is the only platform fulfilling the qualifications and its advantage is the modular concept, so that the data sample is scalable and extendible.

### 6.1.3 The Eclipse Project

eclipse.org [Inc01] is an open source community whose projects are focused on building an integrated and extensible development platform and has over 60 projects. The *Eclipse Project* is the top level and development project dedicated to providing a robust, full-featured, commercial-quality, and freely available industry platform for the development of integrated tools. The data available online at eclipse.org, fulfill all requirements outlined in Section 6.1.1 and has the advantage of being a giant source to extend the analysis with more and different data. Within this thesis we focus on a component of the Platform subproject, besides Equinox (an OSGi framework), JDT (Java development tools) and PDE (plugin development environment) one of the four subprojects of the Eclipse Project. A subproject can again be divided into components. The Platform subproject consists of 13 components. The selected sample data source is the Core component, assuming that it is a central component and therefore the availability and quality of the data is good and the component is in a stable state, thus not being reorganized within the overall project.

### Domain Mapping

The mailing lists, bugzilla categories and plugins at eclipse.org have different granularities. The Eclipse Platform subproject includes 18 mailing lists, 34 different classified bugzilla components and more than 350 plugins. To know which part of the system is affected by a discussion within a mail or problem report discussion is crucial to this analysis. The system with its data sources has to be mapped to cover all sections and classifications. Appendix D shows a complete mapping of the organizational domains coloring the components this thesis is based on.

### Source Code

The source code of eclipse.org is organized modularly.<sup>1</sup> Each plugin contains some functionality and can depend on others. The source code of the Eclipse Platform Core consists of 17 plugins which are qualified in the following Table 6.1. The columns denote the name, the total number of revisions and files, and the timeframe over which modification reports are contributed. The source code is available from the beginning of the development, thus for every file there exists a first revision. The first revision was checked in in May 2001.

<sup>1</sup>Details about eclipse.org and the project's organisation and platform can be found in [Cor06]

Plugin	# Revisions	# Files	Min tstamp	Max tstamp
org.eclipse.core.contenttype	100	49	21.11.2005	03.08.2006
org.eclipse.core.expressions	483	65	19.02.2004	09.01.2007
org.eclipse.core.filesystem	304	60	22.09.2005	19.12.2006
org.eclipse.core.filesystem.hpux.ia64_32	14	7	05.10.2005	03.06.2006
org.eclipse.core.filesystem.hpux.PA_RISC	12	7	05.10.2005	03.06.2006
org.eclipse.core.filesystem.linux.x86	12	6	26.09.2005	03.06.2006
org.eclipse.core.filesystem.linux.x86_64	13	7	26.09.2005	03.06.2006
org.eclipse.core.filesystem.macosx.ppc	7	4	04.10.2005	02.05.2006
org.eclipse.core.filesystem.qnx.x86	10	6	05.10.2005	03.06.2006
org.eclipse.core.filesystem.win32.x86	21	9	26.09.2005	03.06.2006
org.eclipse.core.jobs	160	48	21.11.2005	05.01.2007
org.eclipse.core.resources	3'602	223	02.05.2001	11.01.2007
org.eclipse.core.resources.compatibility	217	65	02.02.2005	03.11.2006
org.eclipse.core.resources.win32	153	30	30.04.2002	08.01.2007
org.eclipse.core.runtime	2'717	149	02.05.2001	29.11.2006
org.eclipse.core.runtime.compatibility	562	60	25.11.2003	15.11.2006
org.eclipse.core.variables	196	32	02.10.2003	26.09.2006
	8'583	827	02.05.2001	11.01.2007

**Table 6.1:** Eclipse Platform Core Plugins

## Mailing List

The mailing list at Eclipse Platform for the Core team is PLATFORM-CORE-DEV@ECLIPSE.ORG. The following quote outlines the official remark on the mailing list homepage for platform-core-dev as of April 2007.<sup>2</sup>

'This is a mailing list where developers of the Platform Core component gather to talk about design issues, code changes or additions, bugs, etc. All Core committers must subscribe to this list. Subscribers to this list also get notices for code changes, build results, testing notices, et cetera. While this list is open to the public, it is intended for discussions related to the implementation of the relevant component. General questions about using Eclipse should be directed to the appropriate Eclipse newsgroup.'

Although the developers are to obliged to subscribe to the list, this is no indication for an active participation as we will see in Section 6.2.1.

## Bugzilla

Reporting a problem concerning an Eclipse Platform plugin supports the allocation of the problem to a component.<sup>3</sup> The two components belonging to the Platform and corresponding to the Core, are Resources and Runtime. The average number of bugs, reported per week and component is 11.6441. This results in an overall number of problem reports of 5331 over the period from October 2001 until January 2007.

The following Section 6.2 shows the classification details of the reported problems for the Platform Resources and Runtime. The first column indicates the classification and the second the occurrence. By looking at the classification of the problem report two aspects arise. First, a lot of

<sup>2</sup>This has been introduced during the realization of this thesis.

<sup>3</sup>The organisation and the mapping of the Eclipse Platform can be found in Table D.1.



reports are later marked as duplicate or invalid. This results in a high number of rejected problem reports. Second, from a point of view interested in the communication between the people, these problem reports are useful because they contain interaction information.

RESOLUTION	#	PRIO.	#	SEVERITY	#	STATUS	#
null	321	P1	119	BLOCKER	209	ASSIGNED	137
DUPLICATE	1'278	P2	208	CRITICAL	356	CLOSED	99
FIXED	1'810	P3	4'907	MAJOR	580	NEW	167
INVALID	964	P4	79	MINOR	104	REOPENED	17
MOVED	-	P5	18	TRIVIAL	37	RESOLVED	4'862
WONTFIX	406			ENHANCEMENT	659	UNCONFIRMED	-
WORKSFORME	448			NORMAL	3'386	VERIFIED	49
REMIND	38						
LATER	66						
	5'331		5'331		5'331		5'331

**Table 6.2:** Bugzilla Report Classification

## 6.2 Project Setup

The integrated database registers 2'615 persons. In order to evaluate the visualization, the circle of people has to be limited. For the following evaluation, we selected the development team that consists of 27 people. The delimitation steps and the time frame alignment are described in Section 4.3.2.

### 6.2.1 Community

The Eclipse Platform Core community compounds of totally 2'615 members within the time period from February 2002 to November 2006. Figure 6.1 shows this composition. The areas represent the team size proportionally. The steel blue colored sector indicates the Bugzilla users, the royal blue the active mailing list subscribers, and the green one the development team. The Bugzilla users form the largest group within the community, with over 2'500 members. The mailing list counts 132 active users and the development team consists of 27 people. While all developer participate in Bugzilla reporting, only the half of the team writes mails to the mailing list. The mail and Bugzilla groups each contain a sub-group composed of members only participating in this group. 100 members contribute actively to more than one group.



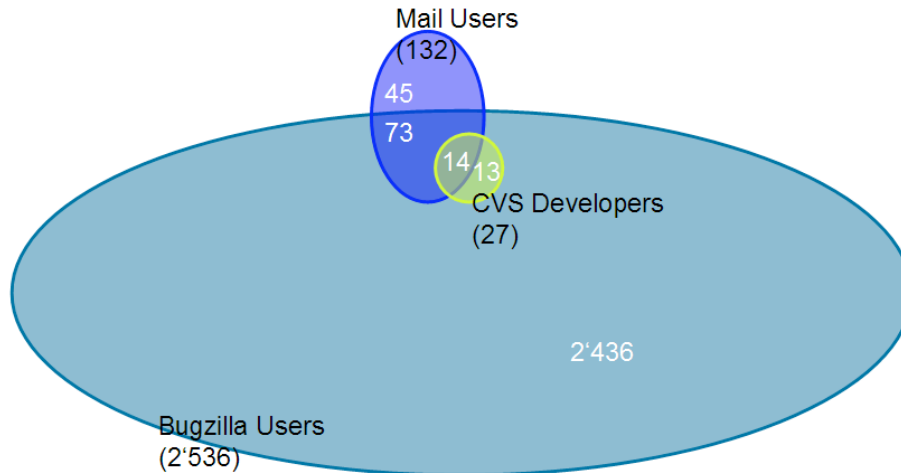


Figure 6.1: Eclipse Platform Core Community

### 6.3 The Eclipse Platform Core Development Team

In order to evaluate the visualization and to analyze the behavioral patterns of the project team, we focus on the core team of the Eclipse Platform Core community. It is illustrated in Figure 6.1. We attempt to answer the questions introduced in Section 3.1 and analyze the core team using the Social Network Analysis Cockpit. We present the results of the scenarios by making snapshots of the graphs. The scenarios are structured as follows: first we address the question that is considered to be answered. Then if patterns are found, a graph depicts the structural overview and the following snippets point them out. The subsequent interpretation of the snippet graphics then explain the findings.

Regarding the time window size, the default values are set to weekly margin, surrounding factor of one and the CVS option activated. This is the standard setup and, if not mentioned, applies to all following evaluations. This means, that the date indicated in the calendar, sets the starting date for the collaboration network period, which lasts one week. The surrounding factor defines the period of the communication network which starts one week earlier and lasts one week longer than the collaboration time frame. Unchecking the CVS check box enables to align the collaboration time frame with the communication time frame.

The following notation is used to describe the analyzed period. The starting date of the collaboration time frame is indicated in the caption of the overview picture. The following snippet pictures are labelled referring to the representation. Details about how to interpret the graphical design is presented in Section 5.4.

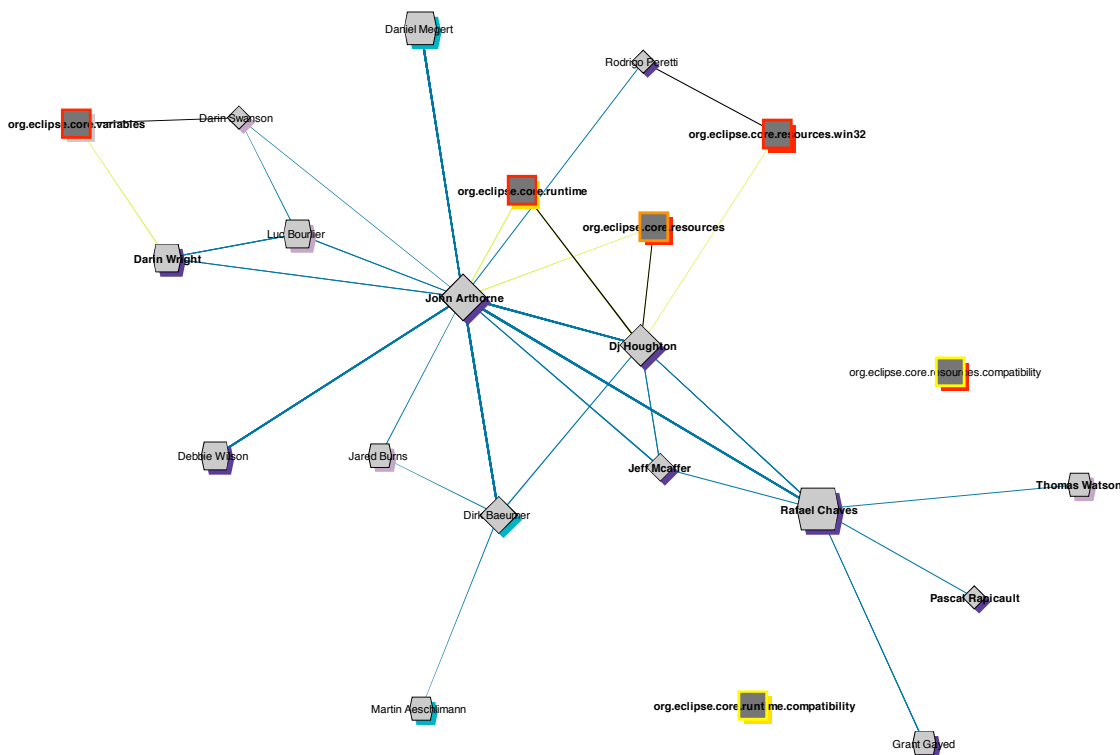
### 6.3.1 Patterns

We first want to introduce patterns that are explored while analyzing the core development team. They will be helpful to address the questions of the next Sections 6.3.2 and 6.3.3.

#### Role: Connector

One of the key roles within a project team is the connector. He knows a lot of team members and is connected to a wide circle of them. The connector is able to bring the right people together either by introducing or interconnecting them. Because he interconnects different groups, he enables the communication or information processing between not directly linked team members. This implies a certain degree of responsibility and leading skills. Such a team member is of value to the project, but incorporates a certain risk. What happens, if a connector leaves the project?

Depending on the team size, it is not reasonable or feasible that everybody is linked to each other. If a team reaches a certain size, there is a need for members who interconnect the groups. The number of connectors within a project team depends on its size and the distribution rate.

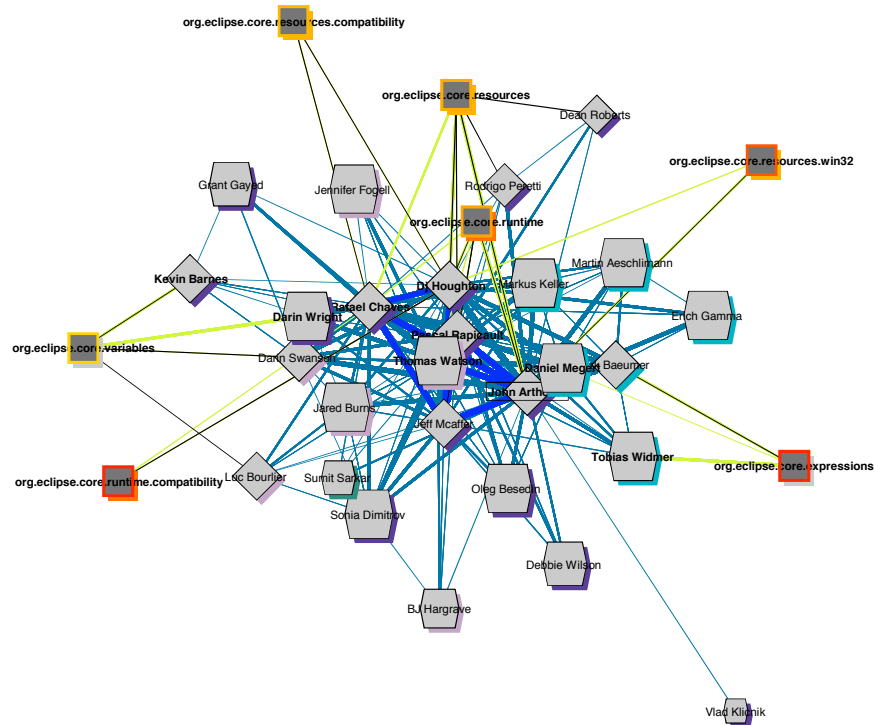


**Figure 6.2:** Connectors of the Eclipse Platform Core Team as of 2003-11-03

Figure 6.2 shows the social network graph of the core development team of Eclipse Platform Core. John and Rafael are such connectors who link groups and team members that otherwise would not be connected. Knowing John can be very helpful for a team member in order to address a question or for the project management that is interested in the status of the project and wants to talk to somebody who is involved in many discussions.

## Role: Communicator

Another key role within a project team is the communicator. This is not necessarily the same person as the connector. A communicator is somebody who knows where the information ideally gets processed through. Hence a communicator can know a connector, but does not need to be one.



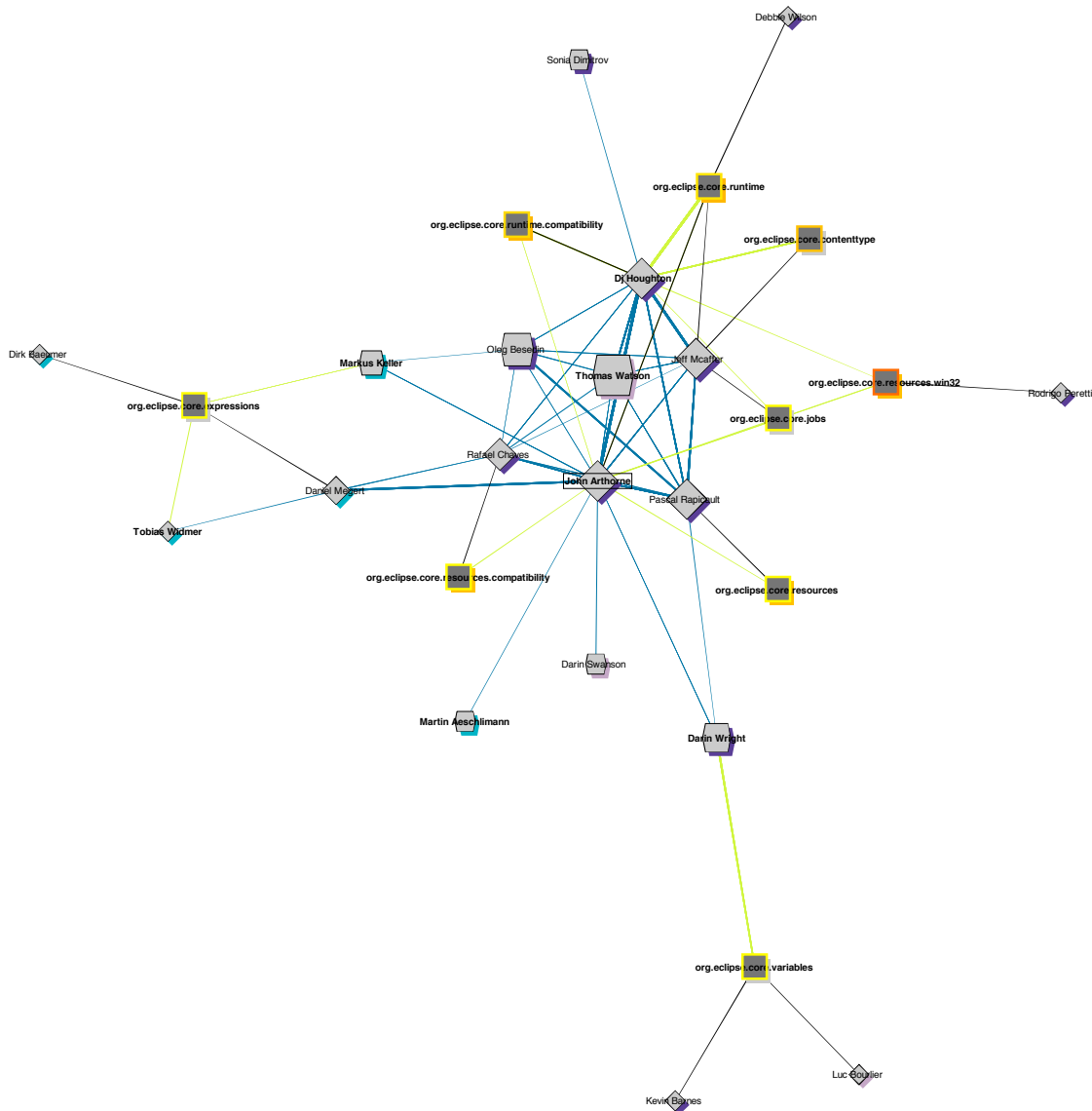
**Figure 6.3:** Communication from May 2004 to February 2006

Figure 6.3 illustrates the communication over 21 months (including only one month of collaboration information). The communication extent is illustrated by the width of the communication path which implies the number of communication paths between these two edges. The more and wider the edges of a person's node are, the more this person communicates.

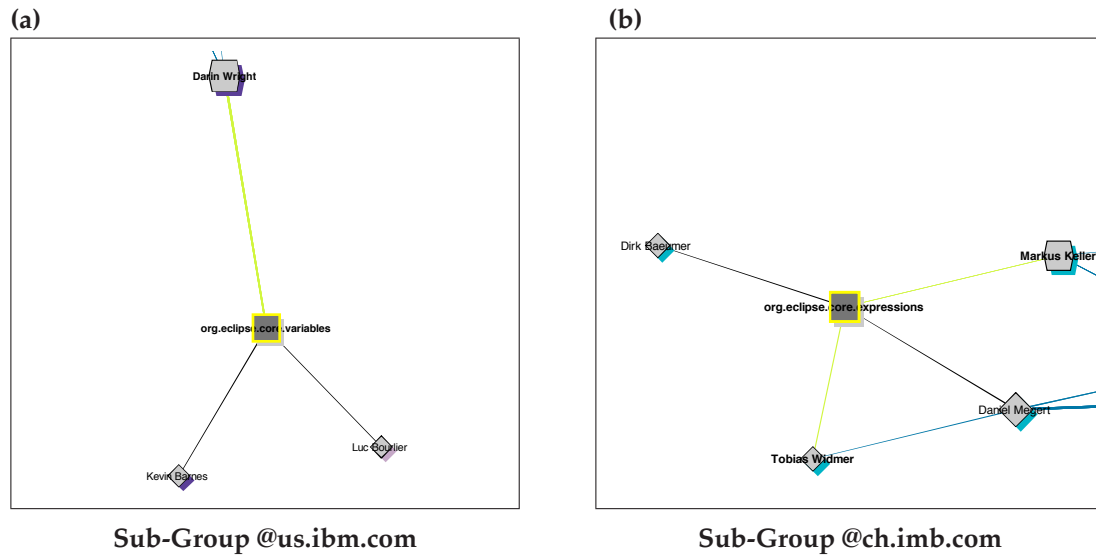
Figure 6.3 shows an interesting aspect that is depicted in the arrangement of the team members. The turquoise colored Swiss members (@ch.ibm.ch) are all aligned on the right side of the network, while the lavender colored group (@us.ibm.com) is located in the left part of the graph. The members discovered at the center of the network all belong to the group @ca.ibm.com. They keep the network together and play an important role within the project such as connector or communicator. They represent the core team of the development team. This is a helpful view for a project manager in order to get an overview of the project's structure. The second insightful finding is the occurrence of mail communication which is illustrated as royal blue interactions. They can solely be found at the center of the network and between the project members located in the very center of the graph. This indicates that the communication with users outside the project team circle, takes place via mailing lists and is performed by only the core project team members.

## Conway's Law

Conway's Law [Con68] states that the governance structure of the project has a direct influence on the structure of the software itself. If this is true, we will discover this pattern in the visualization of the core development team. Given that the team consists of developers either working for IBM United States or IBM Switzerland (except one working for HP). The grouping is based on the email address domain and is outlined in more detail in Section 4.3.1 and Section 5.4.



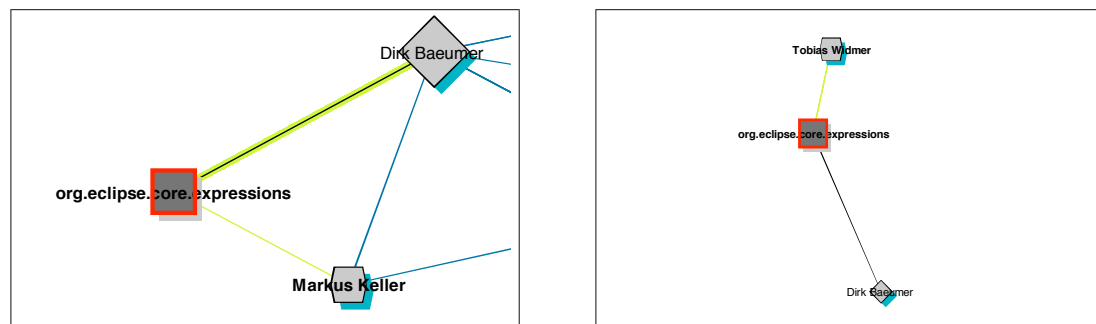
**Figure 6.4:** Conway's Law: Governance Structure Impacts' View as of 2006-04-13



**Figure 6.5:** Snippets of Figure 6.4

Figure 6.5 shows that the organization is reflected in the collaboration structure. Figure 6.5.a illustrates Darin modifying files (indicated by the green line) which he does not own. Sustaining Conway's Law, the owner of the modified files (indicated by the black line) are of the same group, assuming that there is no correlation between @ca.ibm.com and @us.ibm.com, and the location of the team members. Therefore all @ca.ibm.com and @us.ibm.com team members are considered to be IBM United States members. The same pattern applies to the Swiss team. Figure 6.5.b represents the modification reporting of Markus and Tobias. The corresponding file owner of the modified source code are Dirk and Daniel that belong to the same group (indicated by the turquoise node shadow).

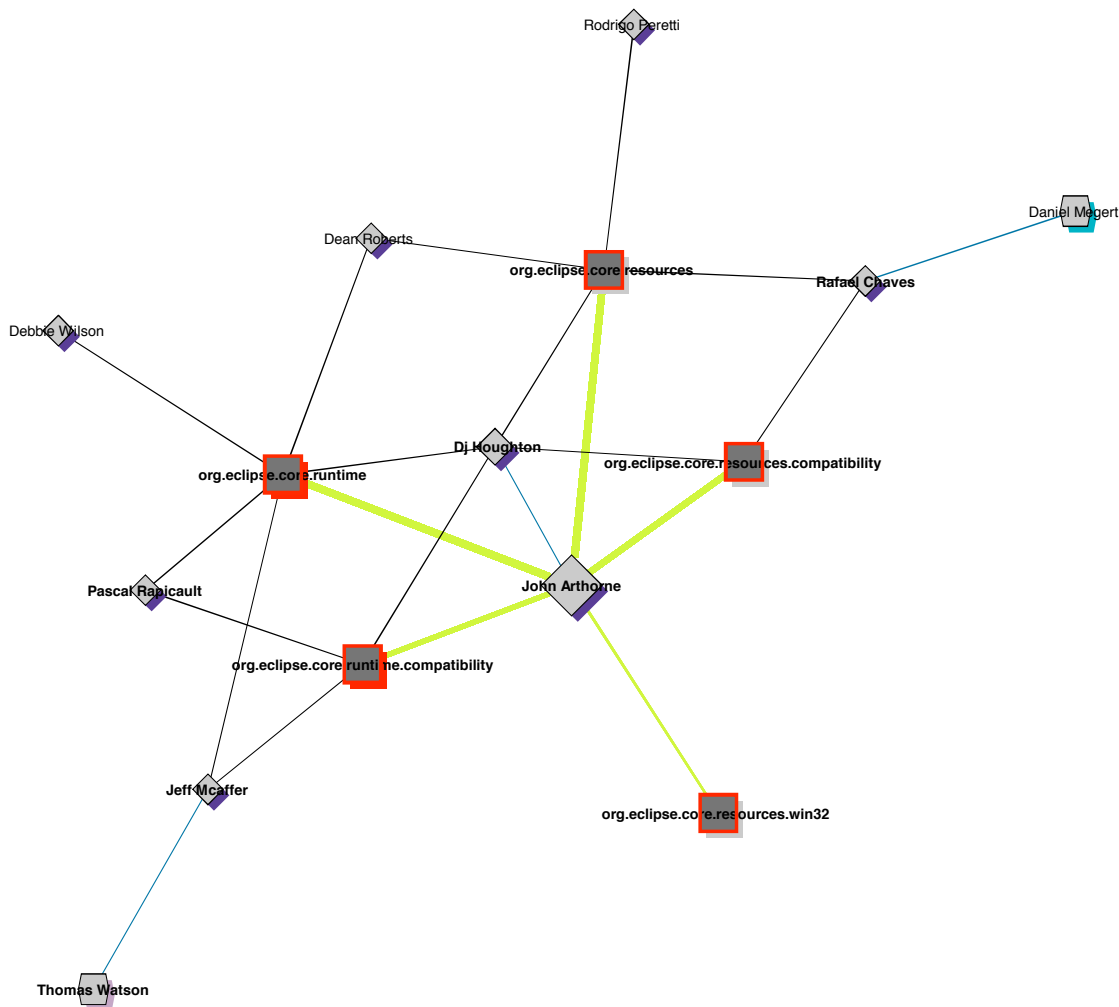
This pattern shows that the closer the people are organized, the more likely they work on the same components. Figure 6.6 shows two more views that illustrate and sustain this pattern.



**Figure 6.6:** More Swiss Contribution Snippets

## Hot Spots

The visualization of the Figure 5.2 introduced in Section 5.1.2 enables to find interesting and active spots in the history of the project regarding source code contribution.<sup>4</sup> We wanted to take a closer look at the interaction network, at a point in time where many modification reports are submitted. Figure 6.7 represents a day at which John updated legal text in the 'about.html' file. The vertical lines in Figure D.1, spanning multiple plugins, indicate clean up operations that affect files of many plugins. A comparison with a larger communication time frame showed that these clean up actions do not correspond to any communication. Therefore, there are only few communication activities occurring within this day. The difference between Figure 6.7 and the Figure 6.8 is the type of the modification report. The first picture shows John modifying files which he does not own whereas in the second picture, the affected files are mainly files he owns.



**Figure 6.7:** Clean Up Revisions at 2005-02-21

<sup>4</sup>The entire overview is illustrated in Appendix D.1.

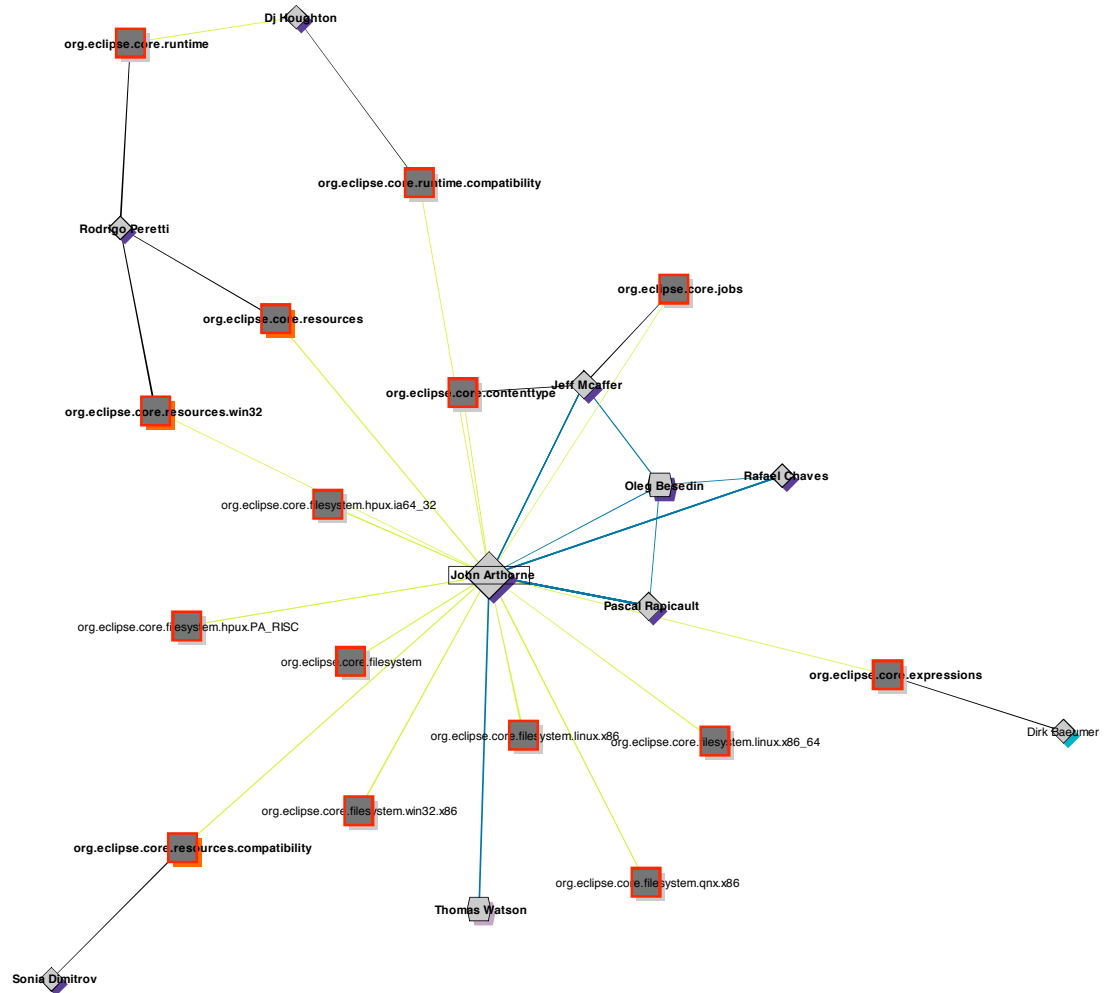


Figure 6.8: Clean Up Revisions at 2006-05-08

### 6.3.2 Perspective: Project Manager

In order to remind the initial questions introduced in Section 3.1, we list them below and outline which of the subsequent scenarios addresses the question or which of the described patterns help to find the answer.

1. *Who is the key personality in respect of communication?*
2. *Who is the leading person regarding code contribution?*
3. *Do sub-groups exist within the project team?*
4. *How are the working teams organized in terms of software development?*

The first two questions can be answered by looking at the graph in Figure 6.9. It shows that John for example is involved in many discussions and DJ is the most contributing source code modification person within the given time frame. The questions number three and four are addressed by the outsourcing scenario. The following sections present two possible scenarios in order to outline the use of the Social Network Analysis Cockpit.

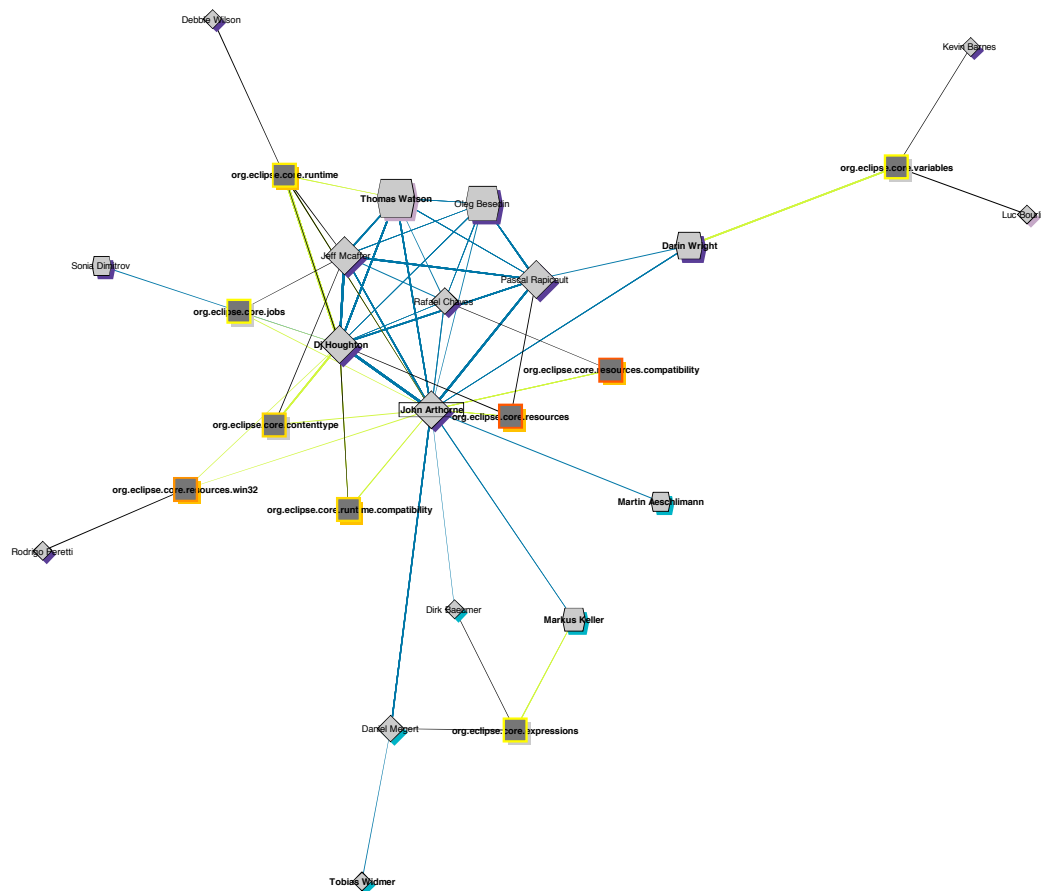
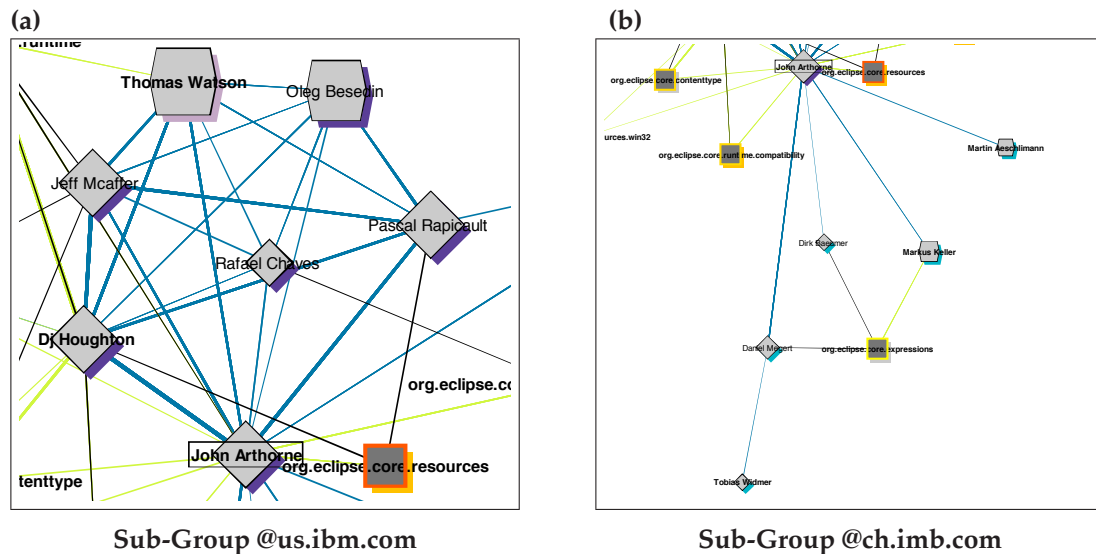


Figure 6.9: A Project Manager's View as of 2006-04-10



## Scenario 1: Outsourcing



**Figure 6.10:** Snippets of Figure 6.9

We reconsider the outsourcing example of Section 3.1.1 with the objective of understanding the collaboration of two geographically distributed teams. Figure 6.9 shows the project setup as of 2006-04-10. During the represented three weeks, there was communication and source code contribution taking place. While looking at the network graph, the two distributed teams can easily be identified. The zoomed snippets are depicted in Figure 6.10. Figure 6.10.a shows the American team and 6.10.b the Swiss team. In addition to the geographical distribution of the teams, the collaboration structure depends on the organisation structure. This phenomenon of 'Conway's Law' was introduced in Section 6.3.1.

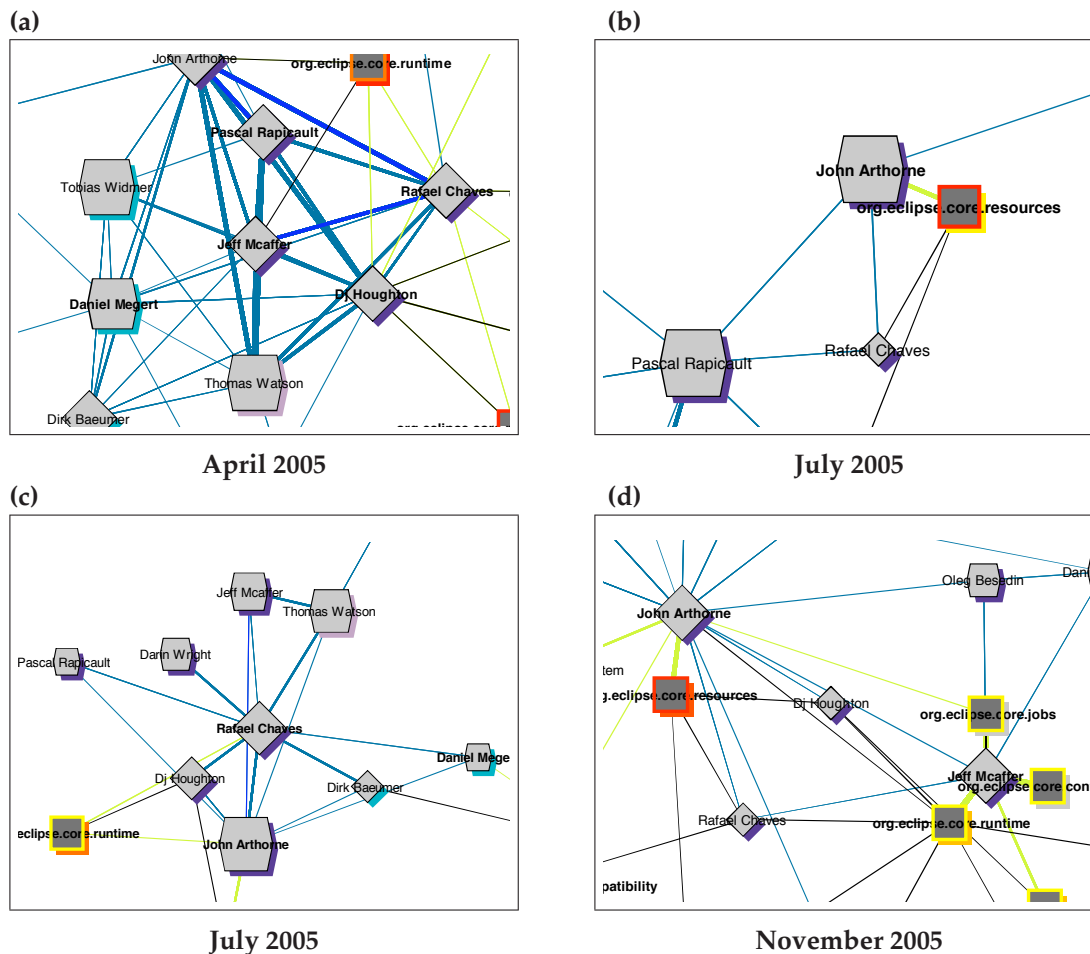
We first take a closer look at the connection point between the teams, in order to understand the collaboration and coordination patterns between the different teams. While looking at the paths between the American and the Swiss members, we see that John is involved in all the communication occurring between the teams. Within the community, this is only a temporal state and does not correspond long-term. If this would be the case in a commercial project, such a SNA Cockpit view would help the project manager to recognize the situation early enough to be able to take action. Otherwise, in case of John being not available, he risks a destabilization of the team structure. This single point of contact constellation can constitute a risk for a project in case this person is not available.

Another interesting aspect is on analyzing the communication behaviour of each of the teams. By looking at the Figure 6.10 the difference is distinctive. While the American members communicate all with each other, the Swiss group depicts only one communication interaction. The reasons can be varying: If members of a team are not located all at the same place, this is an indicator for more tool based communication. Sitting next to each other and talking to each other, as it seem to apply for the Swiss team, this cannot be represented in the graph.

## Scenario 2: Key Person is Leaving the Project

The second scenario presents a key person that is leaving the core development team. The aim is to illustrate the pattern, in order to assist a project manager to recognize the situation early enough to be able to take an action or to assist him with analyzing the new situation and the integration of the potential successor.

Figure 6.11.a depicts a situation in April, where the project setup does not show any irregularities. In July, which is illustrated by Figure 6.11.b, Rafael's contribution to discussions is decreasing until he finishes in contributing his last modification report in July, which is illustrated by Figure 6.11.c. After his final source code contribution, Rafael leaves the core development team. Later he still appears as the owner of source code, but he does not belong to the core development team any more. Although he continues to report problems, which circle he does not leave. As illustrated by Figure 6.11.d, John and Jeff countervail Rafael's departure.

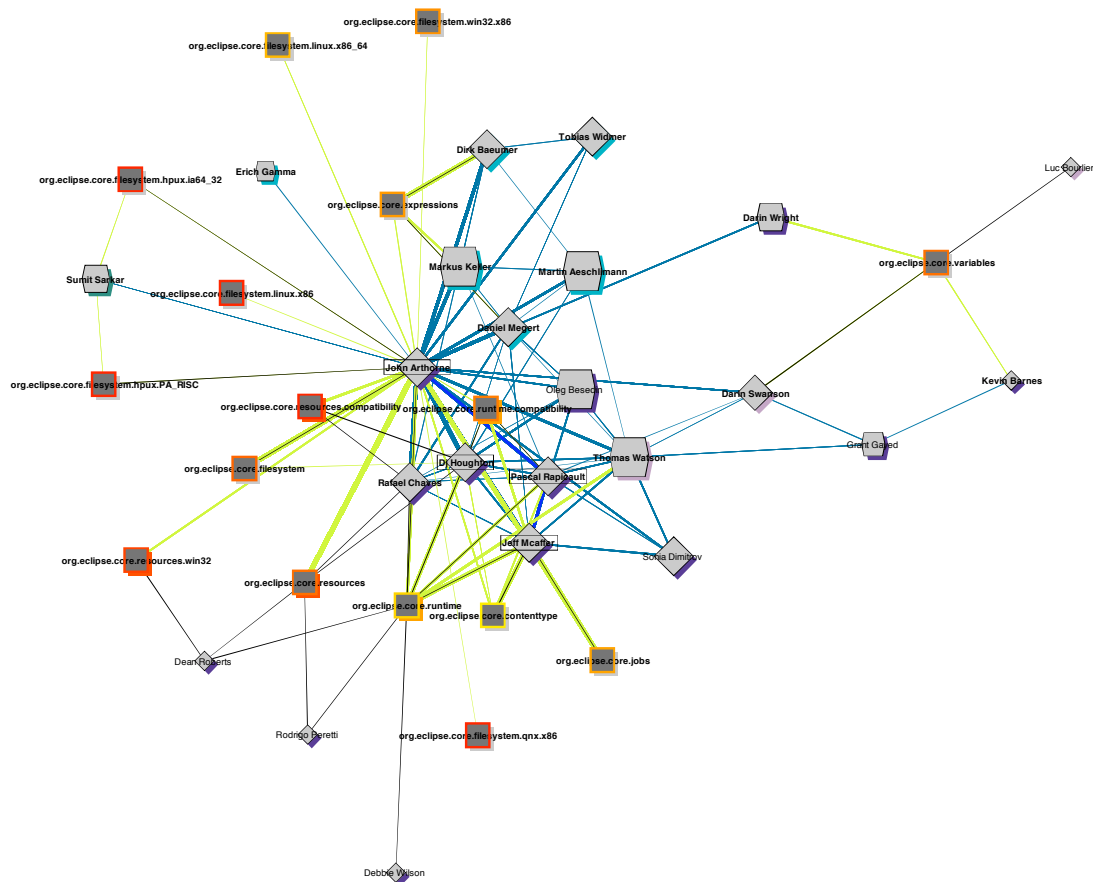


**Figure 6.11:** Rafael Leaving the Core Development Team

### 6.3.3 Perspective: Newcomer

Again, the following list repeats the questions introduced in Section 3.1. This is followed by an illustration that allows the newcomer to answer all these questions. To give further insights into the community, we present a scenario describing the socialization of a core developer.

1. *Who is a leading developer with the best general overview?*
2. *Who is a key personality in terms of software development contribution?*
3. *Who is working on which components?*



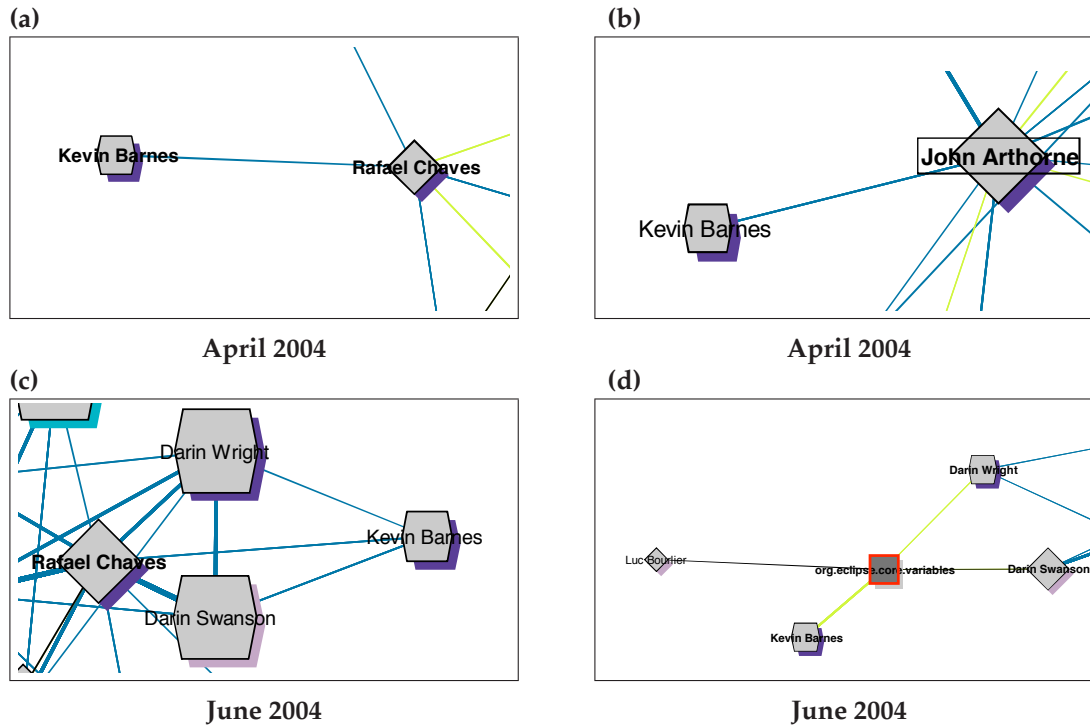
**Figure 6.12:** Collaboration Network as of October 2005

Figure 6.12 shows a five month view on the interaction network. In order to depict the allocation of the developers to the work packages, the time frame of the collaboration network corresponds to the period of the communication network. This results in a representation that gives insight into the development structures and to answer the questions listed above.

Figure 6.12 shows John and Jeff as the most important developers, with John being the leading developer having the broadest overview. The responsible developer or the working team can be discovered for each of the work packages by following the green edges, leading from the work

package to the contributing developer. This helps a newcomer to contact the right people and to get an overview of the project's collaboration.

### Scenario 3: Socialization



**Figure 6.13:** Socialization of Kevin

Figure 6.13 depicts the steps of a successful socialization. The first action of Kevin is to get in touch with Rafael (Figure 6.13.a) and John (Figure 6.13.b). Two months later, communication between Kevin, Rafael, Darin S. and Darin W. (Figure 6.13.c) can be observed. As Figure 6.13.d illustrates, Darin W. is a developer and Darin S. the owner of the files that are going to be modified by Kevin. Rafael plays the role of the connector who introduces the new developer Kevin to the responsible persons for the work scope, Darin W. and Darin S.

This emphasizes the importance of addressing the right people before being allowed to contribute to the software development and being accepted as member of the community.

## 6.4 Summary

Section 6.1 is about the right project selection. The underlying data it is crucial to the outcome of an evaluation. Therefore, it is worth the effort of analyzing different data sources. The following 6.2 describes the selected data set of the Eclipse project. The evaluation Section 6.3 focuses on

answering the questions that arise within projects, by using the Social Network Analysis Cockpit. The first section aims at finding patterns that illustrate different roles or other interesting structures. It demonstrates that the visualization can be used to illustrate various aspects. The following two Sections 6.3.2 and 6.3.3 remind the perspectives that have been introduced in 3.1. Besides answering the questions, scenarios are described in order to discover them in the data set.

Finally, the evaluation shows that the Social Network Analysis Cockpit allows to discover interesting patterns such as the subgrouping of the team. Furthermore, the various aspects that are incorporated within the graphs enables a wide range of different analysis.



# Summary & Conclusion

## 7.1 Summary & Conclusion

This thesis provides a visualization prototype tool, called the Social Network Analysis Cockpit, that allows the project members to interactively explore the project dynamics and shows that different scenarios and roles can be discovered, providing deeper insight into the project. We showed that the history of projects using common software change management tools, such as CVS repositories or Bugzilla databases, contain sufficient information to gain a better understanding of the project.

The evaluation of the Eclipse Platform Core development team sustained that the governance structure of the project has a direct influence on the structure of the software itself and that the visualization enables to identify these structures. Furthermore, we evaluated different roles, such as connectors and communicators and described how to identify them.

By looking at the project from different perspectives, this allows us to state that the modeling of the interaction networks, combining the communication and collaboration networks, is reasonable.

## 7.2 Contribution

We combined communication information, derived from mailing lists and Bugzilla reports, with collaboration information extracted from CVS repositories and built an integrated database. The Social Network Analysis Cockpit is running on this integrated database. The following list presents the contributions this thesis provides.

- A meta model for deriving communication from mail threads and Bugzilla reports.
- Improvements to an existing approach that defines the ownership of a source code file, based on CVS logs.
- Algorithms to obtain an integrated data source combining communication and collaboration information.
- An integrated database containing mail communication, problem report communication, problem reports and modification reports.
- The Social Network Analysis Cockpit for interactive project exploration purposes.

## 7.3 Limitations

Due to the fact that the available data of Open Source Software Projects is hosted on different systems, the extraction and integration is a very time-consuming task. Furthermore, parsing the data from an online view is error-prone in terms of building the object model. Consider a mail thread, where the communication between composers is not modeled explicitly, these missing relations between objects can lead to a high rate of loss during the data extraction.

The wide range of proprietary implementations of social network visualizations does neither allow to process extracted data with existing tools, nor allow to compare the results yet.

## 7.4 Future Work

An integrated system, maintaining communication and collaboration information would facilitate the deployment of such visualization tools and enable to deploy them for example to OSS projects to allow a newcomer to learn something about the project and adapt to the environment and its instruments. It would be interesting to analyze the usage and the user acceptance of such a tool.

The semantic annotation of collaboration information would simplify the data extraction, and due to the additional layer make the implementations more exchangeable.

Visualizing multi dimension data is very challenging, especially when dealing with time-dependent data. A differentiation or the dynamic setting of the abstraction levels would allow to further survey the data. Since the abstraction level of the Social Network Analysis Cockpit is predefined regarding the organization, communication and source code contribution, the visualization only displays a subset of the available data. The integrated database would allow further investigations in this area.

This thesis does not consider any quality measures, such as determination features for the quality of the source code. Including such measurements would allow to make qualified statements about the underlying configuration and direct to the research field of data mining. This would further allow to perform semantic analysis on the data, like for example the survey of the communication content.



# Source Code Ownership

The source code ownership is calculated for every revision of a file. The following section describes the processing and the underlying calculations.

## A.1 Revision Characteristics

Defining ownership requires additional information about a file at a given point in time. The considered features are listed below while the given point in time equals the time stamp of the committing revision. All features are dependent on that time stamp and calculated over the history of the relating file.

- Overall contribution of the committing author
- Owner of the relating file
- Overall contribution of the owning person

### A.1.1 Overall Contribution of the Revision's Author

```
CREATE DEFINER='root'@'localhost' FUNCTION
'getContributionOfAuthor'(p_plugin INT, p_file INT, p_asof TIMESTAMP,
    p_author INT) RETURNS INT
BEGIN
    DECLARE v_contr INT;
    SELECT SUM(linesAdd) - SUM(linesDel) 'contribution' FROM cvs_src.
        revisions r
    WHERE r.file_id = p_file
    AND r.plugin_id = p_plugin
    AND r.author = p_author
    AND r.mr_creation_time <= p_asof
    GROUP BY r.author INTO v_contr;
RETURN v_contr;
```

**Listing A.1:** Function to get the Overall Contribution of the Revision's Author

## A.1.2 Person Owning the Relating File

```

CREATE DEFINER='root'@'localhost' FUNCTION
'getOwner'(p_plugin INT, p_file INT, p_asof TIMESTAMP) RETURNS VARCHAR
(255)
BEGIN
  DECLARE v_owner VARCHAR(255);
  DECLARE v_contr INT;
  SELECT dev.name, max(dev.contribution) FROM (
    SELECT p.name, sum(linesAdd) - sum(linesDel) 'contribution',
      count(*) FROM cvs_src.revisions r
    JOIN cvs_src.author a ON r.author = a.id AND r.plugin_id = a.
      plugin_id
    JOIN bugs.person p ON a.person_id = p.id
    WHERE r.file_id = p_file
    AND r.plugin_id = p_plugin
    AND r.mr_creation_time < p_asof
    GROUP BY p.name) dev
  GROUP BY dev.name
  ORDER BY 2 desc
  LIMIT 1
  INTO v_owner, v_contr;
RETURN v_owner;

```

**Listing A.2:** Function to get the Name of the Person Owning the Relating File

## A.1.3 Overall Contribution of the Owning Person

```

CREATE DEFINER='root'@'localhost' FUNCTION
'getMaxContribution'(p_plugin INT, p_file INT, p_asof TIMESTAMP)
RETURNS INT
BEGIN
  DECLARE v_contr INT;
  SELECT SUM(linesAdd) - SUM(linesDel) 'contribution' FROM cvs_src.
    revisions r
  WHERE r.file_id = p_file
  AND r.plugin_id = p_plugin
  AND r.mr_creation_time < p_asof
  GROUP BY r.author
  ORDER BY 1 DESC
  LIMIT 1
  INTO v_contr;
RETURN v_contr;

```

**Listing A.3:** Function to get the Overall Contribution of the Owner

# Data Preparation

## B.1 Problem Reports

### B.1.1 Bugzilla Importer

The `org.seal.snanalyzer` Eclipse Plug-in contains a bug importer functionality that parses bugs as specified in a list of bugzilla numbers. The underlying java classes are listed below.

- `org.seal.snanalyzer.Activator.java`: Class for the Plug-in,
- `org.seal.snanalyzer.bug.importer.BugParser`: Bugzilla parser to read an eclipse bug in xml format.
- `org.seal.snanalyzer.bug.importer.PersonFinder`: Helper class to find the real name of a person.
- `org.seal.snanalyzer.bug.importer.Util`: Helper class for static conversion implementations for example.
- `org.seal.snanalyzer.bug.importer.mapper.BugModelBuilder`: Builds the object model for the bug report import to store it into the database
- `org.seal.snanalyzer.bug.importer.wizard.BugzillaWizard`:

## B.2 The Integrated Data Model

Figure B.1 shows the integrated data model that contains all classes that are required for the visualization. Both of the integrated database scheme *report.all* and *report.core* contain the tables listed below. The are grouped according to the underlying meta models `org.evolizer.model.versioning`, `org.evolizer.model.issuetracking` and `org.seal.snanalyzer.evaluation.model`.

- `org.evolizer.model.versioning`
  - author
  - revisions
  - source\_units

- `org.evolizer.model.issuetracking`
  - `comment`
  - `component`
  - `issue`
  - `issue_bugactivity`
  - `issue_cc`
  - `mailaddress`
  - `person`
  - `plugin`
- `org.seal.snanalyzer.evaluation.model`
  - `bug_comm`
  - `bug_target`
  - `mail_comm`
  - `mail_target`

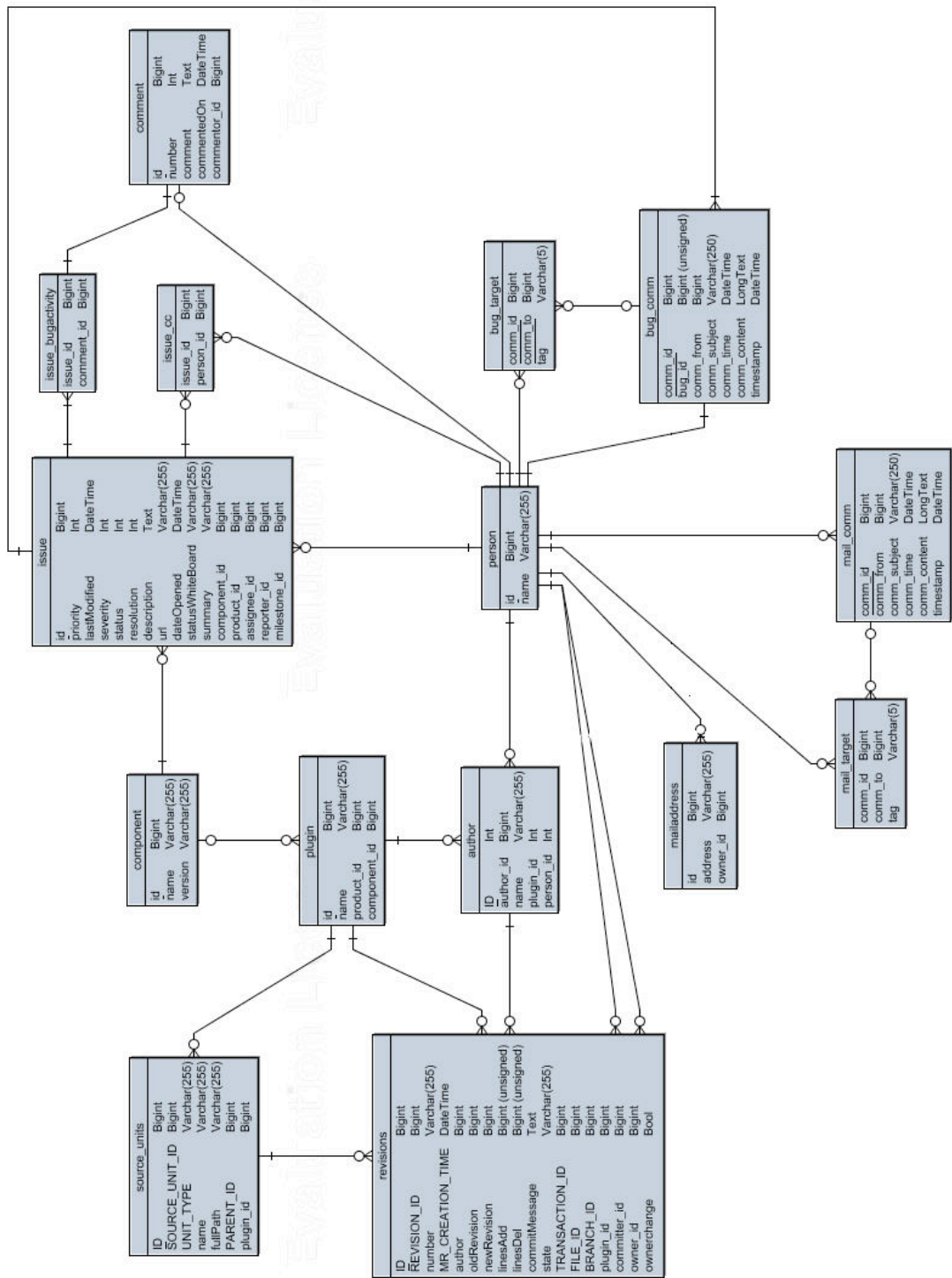


Figure B.1: Detailed Data Model

## B.3 Data Source Consolidation

The integrated data source contains the data of over 2'500 different people while the core development team consists of 27 persons. Since we want to focus on the core team, the integrated data schema is streamlined. The processing steps are listed in the following Table B.1. The integrated schema is named *report\_all* and the consolidated *report\_core*. The table *analysis*, *person\_role\_key* is an analysis of the different roles of the community members. The entire schema *analysis* can be found on the enclosed CD.

Table B.1: Data Source Consolidation :

	report_all		report_core	
1. Modification Reports (order irrelevant)				
plugin	=	17	=	17
source_unit	=	997	=	997
revisions	=	7'479	=	7'479
component	=	3	=	3
2. Person Related Data Extraction				
person	=	2'615	<b>CREATE TABLE person AS</b> <b>SELECT * FROM report_all.person p</b> <b>WHERE EXISTS (</b> <b>SELECT * FROM analysis.person_role_key r</b> <b>WHERE r.commits IS NOT NULL</b> <b>AND p.id = r.id);</b>	27
mailaddress	=	2'681	<b>CREATE TABLE mailaddress AS</b> <b>SELECT * FROM report_all.mailaddress m</b> <b>WHERE EXISTS (</b> <b>SELECT * FROM person p</b> <b>WHERE p.id = m.owner_id);</b>	28
author	=	94	<b>CREATE TABLE author AS</b> <b>SELECT * FROM report_all.author a</b> <b>WHERE EXISTS (</b> <b>SELECT * FROM person p</b> <b>WHERE p.id = a.person_id);</b>	93
3. Communication Data Transformation				
bug_target	=	58'927		

Table B.1: Data Source Consolidation

	report_all		report_core	
	<b>CREATE TABLE</b> bug_target_tmp <b>AS</b> <b>SELECT</b> t.comm_id, t.comm_to, tag <b>FROM</b> bug_target t <b>GROUP BY</b> t.comm_id, t.comm_to, tag;  <b>DELETE FROM</b> bug_target;  <b>ALTER TABLE</b> bug_target MODIFY <b>COLUMN</b> comm_to BIGINT(20) <b>NOT NULL DEFAULT</b> 0, <b>ADD PRIMARY KEY</b> (comm_id, comm_to);  <b>INSERT INTO</b> bug_target <b>SELECT</b> comm_id, comm_to, tag <b>FROM</b> bug_target_tmp t <b>WHERE</b> tag = 'To';  <b>INSERT INTO</b> bug_target <b>SELECT</b> comm_id, comm_to, tag <b>FROM</b> bug_target_tmp t <b>WHERE NOT EXISTS</b> ( <b>SELECT</b> * <b>FROM</b> bug_target f <b>WHERE</b> f.comm_id = t.comm_id <b>AND</b> f.comm_to = t.comm_to);  <b>DROP TABLE</b> bug_target_tmp;	58'863		
		58'682		
			<b>CREATE TABLE</b> bug_target <b>AS</b> <b>SELECT</b> * <b>FROM</b> report_all.bug_target t <b>WHERE EXISTS</b> ( <b>SELECT</b> * <b>FROM</b> person p <b>WHERE</b> p.id = t.comm_to);	22'429

Table B.1: Data Source Consolidation

	report_all		report_core	
bug_comm	=		<b>CREATE TABLE</b> bug_comm <b>AS</b> <b>SELECT * FROM</b> report_all.bug_comm c <b>WHERE EXISTS</b> ( <b>SELECT * FROM</b> bug_target t <b>WHERE</b> t.comm_id = c.comm_id);  <b>DELETE FROM</b> bug_comm <b>WHERE NOT EXISTS</b> ( <b>SELECT * FROM</b> person p <b>WHERE</b> p.id = comm_from);	7'907
mail_target	=	786		
	<b>CREATE TABLE</b> mail_target_tmp <b>AS</b> <b>SELECT</b> t.comm_id, t.comm_to, tag <b>FROM</b> mail_target t <b>GROUP BY</b> t.comm_id, t.comm_to, tag;  <b>DELETE FROM</b> mail_target;  <b>ALTER TABLE</b> mail_target <b>MODIFY COLUMN</b> comm_to BIGINT(20) <b>NOT NULL DEFAULT</b> 0, <b>ADD PRIMARY KEY</b> (comm_id, comm_to);	757		



Table B.1: Data Source Consolidation

	report_all		report_core	
	<b>INSERT INTO</b> mail_target <b>SELECT</b> comm_id, comm_to, tag <b>FROM</b> mail_target_tmp t <b>WHERE</b> tag = 'To';  <b>INSERT INTO</b> mail_target <b>SELECT</b> comm_id, comm_to, tag <b>FROM</b> mail_target_tmp t <b>WHERE NOT EXISTS</b> ( <b>SELECT</b> * <b>FROM</b> mail_target f <b>WHERE</b> f.comm_id = t.comm_id <b>AND</b> f.comm_to = t.comm_to);  <b>DROP TABLE</b> mail_target_tmp;	654		
			<b>CREATE TABLE</b> mail_target <b>AS</b> <b>SELECT</b> * <b>FROM</b> report_all.mail_target t <b>WHERE EXISTS</b> ( <b>SELECT</b> * <b>FROM</b> person p <b>WHERE</b> p.id = t.comm_to);  <b>CREATE TABLE</b> mail_comm <b>AS</b> <b>SELECT</b> * <b>FROM</b> report_all.mail_comm c <b>WHERE EXISTS</b> ( <b>SELECT</b> * <b>FROM</b> mail_target t <b>WHERE</b> t.comm_id = c.comm_id);  <b>DELETE FROM</b> mail_comm <b>WHERE NOT EXISTS</b> ( <b>SELECT</b> * <b>FROM</b> person p <b>WHERE</b> p.id = comm_from);	117
mail_comm	=	891		102



# Social Network Analysis Cockpit Implementation

## C.1 Implementation

### C.1.1 Problem Report Classification

In order to include states of the project into the visualization, we analyze the classification of the reports. The analysis of the classification enables to selected the most significant reports. Table 6.2 shows the classification of the extracted Eclipse Platform Core Bugzilla entries. The priority does not yield any additional information. The severity and the status appear to be promising. A closer look at the resolution shows, that the differentiation of the resolution contains the most informative details about the reports and accordingly to the system.

- NULL → nok (NEW, REOPENED or ASSIGNED reports)
- ~~DUPLICATE~~
- FIXED → ok
- ~~INVALID~~
- ~~MOVED~~
- WONTFIX → ok (sample: [https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=14764](https://bugs.eclipse.org/bugs/show_bug.cgi?id=14764))
- WORKSFORME → ok (sample: [https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=32351](https://bugs.eclipse.org/bugs/show_bug.cgi?id=32351))
- REMIND → ok (sample: [https://bugs.eclipse.org/bugs/show\\_bug.cgi?id=99942](https://bugs.eclipse.org/bugs/show_bug.cgi?id=99942))
- LATER → ok

Duplicate, invalid or moved reports are not of any interest while reports classified as fixed or later state serious reports. The remaining classifications are investigated in order to evaluate their potential consideration. The reports with no resolution classification (NULL) have either status NEW, REOPENED or ASSIGNED. We do not consider these, because it is not yet determined if they are probably a duplicate or an invalid report. The resolution types WONTFIX, WORKSFORME and REMIND have been evaluated by viewing at samples. The discussions within the report do not except any malfunctions. Therefore, these three types are taken into consideration.

### C.1.2 org.seal.snanalyzer.evaluation

The `org.seal.snanalyzer` Eclipse Plug-in contains the Social Network Analysis Cockpit. The underlying java classes are listed below.

- `org.seal.snanalyzer.evaluation.CommunicationModelProvider`: Activates the communication classes.
- `org.seal.snanalyzer.evaluation.DataBaseHandler`: Loads the required database tables.
- `org.seal.snanalyzer.evaluation.ModelContainer`: Stores the objects.
- `org.seal.snanalyzer.evaluation.model.BugComm`: Entity class for bug communication.
- `org.seal.snanalyzer.evaluation.model.MailComm`: Entity class for mail communication.
- `org.seal.snanalyzer.evaluation.views.CollaborationColor`: Color class to define colors for the `SocialNetworkGraph`.
- `org.seal.snanalyzer.evaluation.views.DynamicView`: Loader class for the Social Network Analyzer.
- `org.seal.snanalyzer.evaluation.views.Evolution`: Interface for a (time)scrollable graph.
- `org.seal.snanalyzer.evaluation.views.GraphPanel`: Prototype Application to aligns file revision information.
- `org.seal.snanalyzer.evaluation.views.SampleView`: Plugin starter class that launches the SNA Cockpit.
- `org.seal.snanalyzer.evaluation.views.SNACockpit`: Social Network Analysis Cockpit.
- `org.seal.snanalyzer.evaluation.views.ViewBase`: Superclass of the SNA Cockpit.
- `org.seal.snanalyzer.evaluation.views.graph.PlatformCoreGraph`: The `PlatformCoreGraph` is a special implementation of the `SocialNetworkGraph`.
- `org.seal.snanalyzer.evaluation.views.graph.SampleGraph`: Sample Graph.
- `org.seal.snanalyzer.evaluation.views.graph.SocialNetworkGraph`: The `SocialNetworkGraph` contains a representable yfiles graph structure.

### C.1.3 Test Cases

Test queries for the data fetching.

```
SELECT * FROM report_core.bug_comm m
JOIN bug_target t ON m.comm_id = t.comm_id
WHERE comm_time > '2006-10-29'
AND comm_time < '2006-11-20'
AND m.comm_from != t.comm_to;

SELECT * FROM report_core.mail_comm m
JOIN mail_target t ON m.comm_id = t.comm_id
WHERE comm_time > '2006-10-29'
```

```
AND comm_time < '2006-11-20'
AND m.comm_from != t.comm_to;

SELECT * FROM report_core.revisions
where mr_creation_time > '2006-11-05'
and mr_creation_time < '2006-11-13';

SELECT * FROM report_core.issue
where dateopened > '2006-11-05'
and dateopened < '2006-11-20'
and (resolution is null or resolution in (1, 4, 5, 6));
```

Listing C.1: Test Queries

### C.1.4 Program Start

The following components are required to run the Social Network Analysis Cockpit.

- Modified Revision 336 of org.evolizer.model.issuetracking
- Modified Revision 336 of org.evolizer.model.versioning
- org.seal.snalyzer

The program is launched by clicking Window/Show View in the workspace of the Eclipse IDE, and selecting the feature indicated in Figure C.1

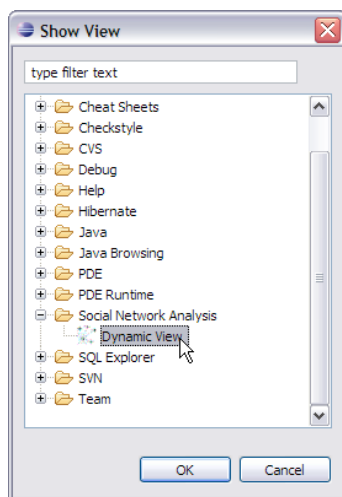


Figure C.1: Starting the Eclipse Plug-in View

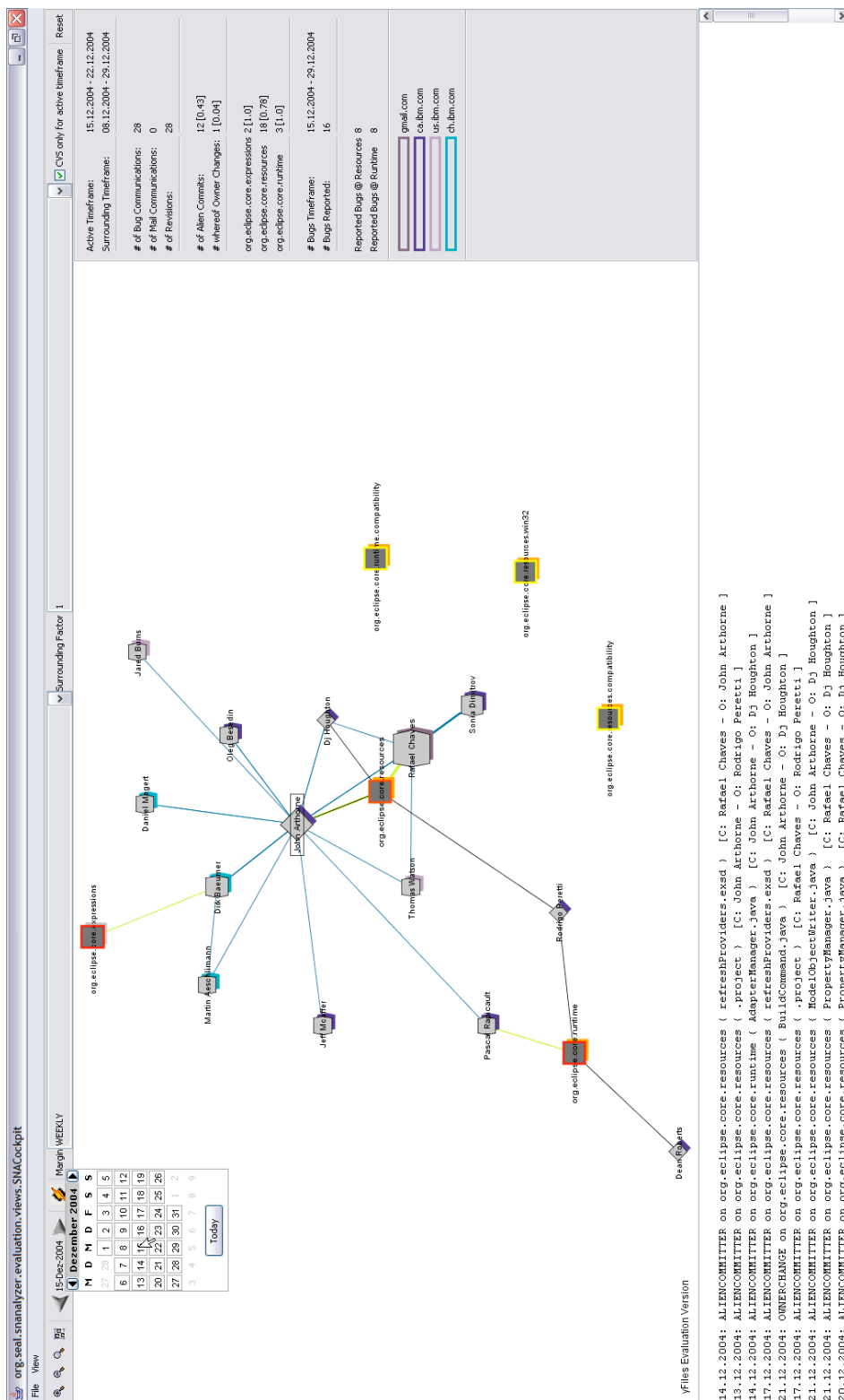


Figure C.2: Social Network Analysis Cockpit

## Appendix D

---

# The Eclipse Project

Name	Plug-ins	Mailing List	Bugzilla
Platform.Ant	org.eclipse.ant.*	platform-ant-dev	Platform.Ant
	org.eclipse.ui.externaltools		
Platform.Core	org.eclipse.core.contenttype	platform-core-dev	Platform.Runtime
	org.eclipse.core.expressions		Platform.Resources
	org.eclipse.core.filesystem.*		
	org.eclipse.core.jobs		
	org.eclipse.core.resources.*		
	org.eclipse.core.runtime.*		
	org.eclipse.core.variables		
Platform.CSV	org.eclipse.team.cvs.*	platform-cvs-dev	Platform.CVS
Platform.Debug	org.eclipse.debug.*	platform-debug-dev	Platform.Debug
Platform.Releng	org.eclipse.releng.*	platform-releng-dev	Platform.Releng
Platform.Search	org.eclipse.search	platform-search-dev	Platform.Search
Platform.SWT	org.eclipse.team.swt	platform-swt-dev	Platform.SWT
Platform.Team/Compare	org.eclipse.team.*	platform-team-dev	Platform.Team
	org.eclipse.compare.*	platform-compare-dev	Platform.Compare
Platform.Text	org.eclipse.text	platform-text-dev	Platform.Text
	org.eclipse.core.filebuffers		
	org.eclipse.jface.text		
	org.eclipse.workbench.texteditor		
	org.eclipse.ui.editors		
Platform.UserAssistance	org.eclipse.help.*	platform-ua-dev	Platform.UserAssistance
	org.eclipse.ui.intro.*		
	org.apache.lucene		
	org.eclipse.tomcat		
	org.eclipse.ui.cheatsheets		
	org.eclipse.ui.browser		
Platform.UI	org.eclipse.ui.*	platform-ui-dev	Platform.UI
	org.eclipse.jface		
Platform.Update	org.eclipse.update.*	platform-update-dev	Platform.Update
Platform.WebDAV	org.eclipse.webdav	platform-webdav-dev	Platform.WebDAV
Equinox	org.eclipse.equinox.*	equinox-dev	Equinox.Budles
	org.eclipse.osgi.*		Equinox.Framework
JDT.APT	org.eclipse.jdt.apr.*	jdt-apt-dev	JDT.APT
JDT.Core	org.eclipse.jdt.core	jdt-core-dev	JDT.Core
JDT.Debug	org.eclipse.jdt.debug.*	jdt-debug-dev	JDT.Debug
	org.eclipse.jdt.launching		
JDT.UI	org.eclipse.jdt.ui.*	jdt-ui-dev	JDT.UI
	org.eclipse.ltk.core.manipulation		
	org.eclipse.ltk.core.refactoring		
	org.eclipse.jdt.junit.*		
	org.eclipse.jdt.doc.user		
PDE.Build	org.eclipse.pde.build	pde-build-dev	PDE.Build
PDE.UI	org.eclipse.pde.ui.*	pde-ui-dev	PDE.UI

**Table D.1:** Eclipse Platform Organization





Figure D.1: Contribution to Source Code Development over Time



# Content of CD-ROM

- 01 Data Retrieval
  - Database (bugs, cvs\_core\*, mail\_threads)
  - org.evolizer.base (CVSParser, modified)
  - org.evolizer.model.issuetracking (modified)
- 02 Data Cleaning
  - Database (bugs, cvs\_src, mails)
  - SQL Functions
- 03 Analysis
  - Database (report\_all, report\_core, analysis, util)
  - org.evolizer.model.issuetracking (modified)
  - org.evolizer.model.versioning (modified)
  - org.seal.snanalyzer



---

# References

- [Abe07] Mark Aberdour. Achieving quality in open source software. *IEEE Software*, pages 59–65, 2007.
- [BH98] Ivan T. Bowman and Richard C. Holt. Software Architecture Recovery Using Conway’s Law. In *Proceedings of the 1998 Conference of the Centre for Advanced Studies on Collaborative research*, page 6. CASCAN, IBM Press, 1998.
- [CO03] Kevin Crowston and Charles S. Osborn. A Coordination Theory Approach to Process Description and Redesign. Working papers WP 4029-98. CCSTR ; 204., Massachusetts Institute of Technology (MIT), Sloan School of Management, April 2003.
- [Con68] Mel Conway. How Do Committees Invent? *Datamation Magazine*, April 1968.
- [Cor06] International Business Machines Corp. Eclipse platform technical overview. <http://www.eclipse.org/articles/whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf>. Last visited in March 2007, 2006.
- [Cro97] Kevin Crowston. A Coordination Theory Approach to Organizational Process Design. *Organization Science*, 8(2):157 – 175, March 1997.
- [CWL<sup>+</sup>05] Kevin Crowston, Kangning Wei, Qing Li, U. Yeliz Eseryel, and James Howison. Coordination of Free/Libre Open Source Software Development. In *Proceeding of the 26th International Conference on Information Systems (ICIS)*, Las Vegas, USA, December 2005. International Conference on Information Systems (ICIS).
- [Duc05] Nicolas Ducheneaut. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323 – 368, August 2005.
- [FLO04] FLOSSmole. Collaborative Collection and Analysis of Free/Libre/Open Source Project Data. <http://ossmole.sourceforge.net/index.html>. Last visited in April 2007, since 2004.
- [Fou98] Mozilla Foundation. Mozilla - Preserve Choice and Innovation on the Internet. <http://www.mozilla.org/>. Last visited in April 2007, since 1998.
- [FPG03] Michael Fischer, Martin Pinzger, and Harald Gall. Populating a Release History Database from Version Control and Bug Tracking Systems. In *Proceedings of the 19th International Conference on Software Maintenance (ICSM)*, pages 23 – 32, Amsterdam, The Netherlands, September 2003. IEEE, IEEE Computer Society.

- [GKSD05] Tudor Grba, Adrian Kuhn, Mauricio Seeberger, and Stphane Ducasse, editors. *How Developers Drive Software Evolution*. Eighth International Workshop on Principles of Software Evolution (IWPSE), 2005.
- [Gla02] Malcolm Gladwell. *The Tipping Point: How Little Things Can Make a Big Difference*. Back Bay Books, January 2002.
- [Glo] Peter A. Gloor. Temporal Communication Flow Visualizer. <http://www.swarmcreativity.net>, <http://www.soberit.hut.fi/T-76.5651/>, <http://www.ickn.org/>. Last visited in April 2007.
- [HG99] James D. Herbsleb and Rebecca E. Grinter. Splitting the Organization and Integrating the Code: Conway’s Law Revisited. In *Proceedings of the 21st international conference on Software engineering (ICSE)*, pages 85 – 95, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.
- [HIC06] James Howison, Keisuke Inoue, and Kevin Crowston. Social Dynamics of Free and Open Source Team Communication. In *Proceedings of the IFIP 2nd International Conference on Open Source Software (IFIP)*, Lake Como, Italy, June 2006. International Federation for Information Processing (IFIP).
- [IBM] IBM. Many eyes. <http://services.alphaworks.ibm.com/manyeyes/home>.
- [Inc] Red Hat Inc. Hibernate - Relational Persistence for Java and .NET. <http://www.hibernate.org/>. Last visited in April 2007.
- [Inc01] Eclipse Foundation Inc. Eclipse - an Open Development Platform. <http://www.eclipse.org/>. Last visited in March 2007, since 2001.
- [JH04] Kevin Crowston James Howison. The Perils and Pitfalls of Mining SourceForge. In *Workshop on Mining Software Repositories*. International Conference on Software Engineering ICSE, 2004.
- [MH094] MHonArc. A Mail-to-HTML Converter. <http://www.mhonarc.org>. Last visited in April 2007, since 1994.
- [MO07] Christian Bird Premkumar Devanbu Alex Gourley Michael Ogawa, Kwan-Liu Ma. Visualizing social interaction in open source software projects. In *APVIS ’07. 2007 6th International Asia-Pacific Symposium on Visualization*, pages 25–32, Feb. 2007.
- [OST04] Inc. (Open Source Technology Group) OSTG. SourceForge<sup>®</sup> Collaborative Development Environment. <http://sourceforge.net/>. Last visited in April 2007, since 2004.
- [PtFSF98] GNU Project and the Free Software Foundation. CVS - Concurrent Versions System. <http://www.nongnu.org/cvs/>. Last visited in April 2007, since 1998.
- [Sac01] Warren Sack. Conversation Map: An Interface for Very Large-Scale Conversations. *Journal of Management Information Systems*, 17(3):73 – 92, 2001.
- [SDD<sup>+</sup>06] Warren Sack, Franoise Dtienne, Nicolas Ducheneaut, Jean-Marie Burkhardt, Dilan Mahendran, and Flore Barcellini. A Methodological Framework for Socio-Cognitive Analysis of Collaborative Design of Open Source Software. *Computer Supported Cooperative Work (CSCW)*, 15(2):229 – 250, June 2006.

- [SEA04] SEAL. Software Evolution and Architecture Lab. <http://seal.ifi.unizh.ch/>. Last visited in April 2007, since 2004.
- [WBB76] Harrison C. White, Scott A. Boorman, and Ronald L. Breiger. Social Structure from Multiple Networks. i. Blockmodels of Roles and Positions. *The American Journal of Sociology*, 81(4):730 – 780, jan 1976.
- [XCM05] Jin Xu, Scott Christley, and Gregory Madey. The Open Source Software Community Structure. In *Proceeding of the Annual North American Association for Computational Social and Organizational Science (NAACSOS)*, Notre Dame, IN, USA, June 2005. North American Association for Computational Social and Organizational Science (NAACSOS).
- [XGCM05] Jin Xu, Yongqin Gao, Scott Christley, and Gregory Madey. A Topological Analysis of the Open Souce Software Development Community. In *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS)*, page 198, Washington, DC, USA, January 2005. IEEE Computer Society.
- [yG] yWorks GmbH. yfiles - Extensive Java Library Providing Algorithms and Enabling the Visualization of Graphs. <http://www.yworks.com>. Last visited in April 2007.