Process Mining Software Repositories

Wouter Poncin
August 2010
Master's Thesis

Process mining software repositories

by

Wouter Poncin

Supervisor

dr. A. Serebrenik
dr.ir. B.F. van Dongen

Eindhoven, August 20, 2010
Acknowledgements

This thesis is the result of my graduation project which completes my master Computer Science & Engineering at the Eindhoven University of Technology. The project was performed internally at the Software Engineering & Technology group in collaboration with the Architecture of Information Systems group at the Mathematics and Computer Science department of the Eindhoven University of Technology.

First of all I would like to thank Alexander Serebrenik for his guidance and feedback during the project, as well as during the seminar Software Engineering & Technology, where I defined the outline and goals for this project. I would also like to thank Boudewijn van Dongen for his feedback during the project.

Furthermore, I would like to thank the students of the Software Evolution course for their feedback on the first versions of FRASR. I also would like to thank the LaQuSo staff and ProM developers for their interest and feedback regarding FRASR.

WOUTER PONCIN

Eindhoven
August 20, 2010
Abstract

Software developers use several software repositories (like version control systems, bug trackers and mailing list archives) to share, communicate and backup information related to a software project. Analyzing data from software repositories can help to improve the software development process, which is one of the factors which influences the success of a project. Furthermore, software repositories allow analysis of the development process, without interaction with the developers involved.

Current applications for analyzing data from software repositories mainly focus on a single type of software repository. Other applications which do have support for various types of software repositories typically focus only on a specific analysis aspect, like calculating metrics, and ignores temporal relations between events (like reporting a bug, creating a revision, etcetera). Temporal relations between events contain valuable information about the development process. Process mining focuses on those temporal relations. Process mining has been demonstrated as a valuable technique for analyzing processes in various business domains. Currently, no applications can be used to apply process mining techniques to the combination of data from various software repositories.

To solve this problem, an application called FRASR (FRamework for Analyzing Software Repositories), has been developed. FRASR can be used to analyze the development process of software projects, using data from software repositories. FRASR allows a user to export data from several software repositories (identified by a URL for example) to an event log. This log can then be used by analysis applications, such as ProM (process mining workbench). As developers use various aliases in the software repositories, FRASR supports automatic matching of those aliases. This matching is necessary since the aliases provide the only useful way of linking data between the repositories.

FRASR has been successfully used in two series of case studies, demonstrating the applicability of process mining to analysis questions related to the development process (without consulting the people involved), and the added value of using the combination of data from multiple software repositories, as opposed to using the data from multiple software repositories in isolation.

Keywords: case definition, event log, process mining, software engineering, software repositories

Classification: D.2.9 [Software engineering]: Management, H.2.8 [Database management]: Database applications—Data mining, K.6.3 [Management of computing and information systems]: Software Management—Software process
Contents

Acknowledgements i

Abstract iii

1 Introduction 1

2 Preliminaries 3
  2.1 Software repositories 3
  2.2 Process mining 6
  2.3 Summary 11

3 Requirements 13
  3.1 Facilitate process analysis 13
  3.2 Various software repositories 14
  3.3 Easy manner 14
  3.4 Summary 15

4 FRASR Design 17
  4.1 Object model 17
  4.2 Developer matching 25

5 FRASR Implementation 31
  5.1 Cache 31
  5.2 GUI 32
  5.3 External libraries 32

6 Developer Matching Experiments 35

7 Case Studies 39
  7.1 Case study 1: TU/e software engineering project 40
  7.2 Case study 2: Open source software development 50
  7.3 Conclusion 60

8 Related Work 61
  8.1 Single software repository 61
  8.2 Multiple software repositories 64
  8.3 Shortcomings 67
Chapter 1

Introduction

Software development is an activity which is mostly performed in teams. Due to the size, complexity and time-to-market constraints of modern software systems, it is infeasible that such systems are developed by a single developer.

Software development often takes place in a geographically distributed team. Consider for example open source or outsourcing projects. Developers in such projects are forced to use auxiliary applications for communication and coordination on the development effort. Examples of such applications are mailing lists, bug trackers and version control systems. Mailing lists provide communication means between developers, bug trackers and version control systems provide means for development effort coordination. All such applications produce archives reflecting the development process. These archives are known as software repositories.

Analyzing the development process of software projects is interesting, since the development process is one of the factors influencing the projects success [Jia04]. Furthermore, performing this analysis without having to consult the people involved removes the need to schedule meetings with those people for instance, leaving more time for them to work on the project. An other reason is that the knowledge of the people involved can be limited. In that case, analyzing the software repositories can provide new insights. However, analyzing the development process without having to consult the people involved is hard, as developers use several software repositories to store historic data related to the development process.

Current applications for analyzing data from software repositories mainly focus on a single type of software repository. As the development process involves several software repositories, these applications are not usable for this purpose. However, it is not always the case that the data from all the repositories is necessary to answer a specific analysis question. In that case, these applications are still usable. Other applications for analyzing various types of software repositories, typically focus only on a specific analysis aspect, like calculating metrics. See Chapter 8 for a more detailed description of the current applications.

In order to answer an analysis question like 'has the V-model for software development been applied correctly' [Moo01], techniques which focus on the temporal relations between events are required. The V-model states that each phase of development should be completed, before the next phase is initiated. Currently no applications exist supporting these kinds of techniques.

As process mining focusses on temporal relations between data, it is well suited for analyzing the data from software repositories, used in the development process of software projects. Furthermore, it has been demonstrated that process mining is a valuable technique for ana-
Analyzing business processes in various domains [Roz09, Aal07b, Ude08, Gün07, Son09].

As there is currently no application available for analyzing data from various software repositories using process mining techniques, we define the following project goal:

**Goal of the project:** Create an application which facilitates process analysis of data from various software repositories, in an easy manner.

By facilitate we mean that the application does not have to provide analysis algorithms and visualizations for data from the software repositories, as such applications already exist, e.g. the ProM process mining workbench [Don05b, Aal07a]. ProM is an open source platform independent application which contains more than 280 mining, visualization and conversion algorithms. In order to use an analysis application like ProM, the application to be developed should provide means to export the data from the software repositories to a format which can be used by ProM. By ‘in an easy manner’, we mean that the user of the application does not have to do the processing of data from the data sources by hand, but only has to supply the location at which the data is stored (by providing a URL, for example). See Chapter 3 for further details about the requirements the application should meet.

The application implemented for this project is called FRASR (FRamework for Analyzing Software Repositories). The key features of FRASR are:

- FRASR supports various commonly used software repositories, like Subversion [Sub09], Bugzilla [Bug09], Trac [Tra09] and SourceForge [Sou10]. A complete list of supported software repositories can be found in Chapter 4.
- FRASR allows the user to define the way the data from the software repositories is mapped onto the format used to export the data. This mapping is important as it influences the possible analysis techniques applicable to the data.
- FRASR can automatically construct the matching of developer aliases (used in the software repositories). This is necessary since developers use different aliases in the various software repositories, and the aliases contain the only useful information to link data from various software repositories (See Chapter 6 for more information).
- FRASR is easily extendable, which allows a user to add support for new software repositories and analysis applications. Appendix D describes the ways in which FRASR can be extended.

Chapter 7 describes two case studies which have been carried out using FRASR and ProM. In the first case study, we demonstrate the applicability of process mining techniques to data from software repositories. We show that we can get insights in the development process of a software project, without having to contact the developers involved in the project. In the second case study, we show the added value of using data from multiple software repositories in combination, as opposed to using data from multiple software repositories in isolation.
Chapter 2

Preliminaries

This thesis discusses the application of process mining techniques to software repositories, and as such builds upon the existing results in these areas. In this chapter we introduce key notions pertaining to software repositories (Section 2.1) and process mining (Section 2.2). We postpone a more thorough review of the related work to Chapter 8.

2.1 Software repositories

In this thesis, software repositories are defined as repositories where data from the development of a software project is stored [Rub07]. These repositories can be used to share information, to backup and retrieve information or to communicate information to other developers.

Several different types of software repositories have been developed and are currently used by software developers. Version control systems are mainly used to store the source files, bug trackers contain the information about reported problems and mailing lists contain information about the communication between developers. These and other types of software repositories are described in more detail in the next sections.

2.1.1 Version control systems

Version control systems contain multiple versions (revisions) of the same unit of information (files for example). Using a version control system allows developers to work on the same piece(s) of text, usually source code, simultaneously. The version control system takes care of combining (merging) the modifications of all the developers. Furthermore, version control systems can be used to create backups.

Version control systems come in two categories: centralized and decentralized [Bet10]. Centralized version control systems are characterized by the presence of one central repository, accessible for the developers. The developers have a local copy of the (entire) repository. After making modifications to this local copy, the changes are send (committed) to the central repository. Figure 2.1 presents an example of the use of a centralized version control system. The blue rectangles represent a version of an artifact (labeled with r1 - r4), the other rectangles represent the changes between the versions and the circles represent the developers making the changes.

An example of a centralized version control system is Subversion [Sub09]. Listing 2.1 shows the Subversion log of revision 13490 of the ArgoUML project [Arg10]. This revision, created
Figure 2.1: The addition of information to an artifact in a centralized version control system [Bet10].

by user bobtarling at 2007-09-05 00:54:04 +0200, contains a move (lines 5 and 9), a copy +
manual delete (lines 3 and 4), a copy (line 6) and three modifications (lines 7, 8 and 10). Line
12 contains the comment, provided by the user.

Listing 2.1: Subversion log of the ArgoUML project [Arg10]

Having one central repository allows for easy analysis of the data from the project as it
is all contained at a single location. However, a central repository implies a single point of
failure. When the central repository is not accessible (due to network problems for example),
the developers can only work in their local repositories, and hence cannot combine their work
with the work of the other developers.

Solving this problem was one of the goals of decentralized version control systems. In these
systems, every developer has a local repository, containing all the revisions, to which the
modifications are committed. To share data, a developer can send and receive modifications
from an other developer. This approach eliminates the single point of failure. However, when
the personal repository of a developer gets corrupted, there is no backup at the a central
server. Furthermore, for an analist it is much harder to retrieve all the modifications, as these
are not available at a single location.

Several version control systems have been developed and used for software development.
CVS (Concurrent Versions System) [Cvs09] and Subversion (SVN) [Sub09] are examples of
centralized version control systems. Examples of decentralized version control systems are
Git [Git10], Bazaar [Baz10] and Mercurial [Mer10]. The difference between CVS and Sub-
version is that subversion records changes to multiple artifacts per revision, where CVS only
2.1. Software repositories

records changes per artifact.

2.1.2 Bug trackers

During the development of a software project, developers often use a bug tracker to report issues, feature requests, etcetera (called bug reports). A bug report typically contains several properties (supplied by the developers) like the status (new, assigned, resolved, closed, etcetera), the resolution (fixed, invalid, duplicate, etcetera) and the severity (blocker, major, minor, trivial, etcetera). Developers can post comments to the bug reports to share their findings. Bug reports are often stored in a central repository. Using this repository, developers can query for known issues concerning a software component for example. Furthermore, quality assurance teams and project managers can generate reports to get insight in the progress of the development. Figure 2.2 presents an example of the lifecycle of a bug report in a Bugzilla [Bug09] repository. Each box represents a possible state and each arrow represents a change from one state to another.

Examples of bug tracking systems are: Bugzilla [Bug09], Mantis Bug Tracker [Man10] and Launchpad Bugs [Lau10]. The Trac system [Tra09] also contains issue tracking facilities.

2.1.3 Mailing lists

An important aspect of developing software in a team, is communication. Specially in geographically distributed software development teams, mailing lists are often used as a communication medium. Mailing lists are used to send messages (email) to a group of developers (who have subscribed to the list). Before the existence of version control systems, mailing lists were also often used to distribute patches. Nowadays, this approach is still used by developers.
who do not have write access to the version control system. Besides the messages delivered to the subscribers, copies of the messages are often stored at a repository, to provide a searchable archive of all the messages (consider the MARC mail archive for example [Mar10a]).

### 2.1.4 Other software repositories

Besides version control systems, bug trackers and mailing lists, also other software repositories are gaining popularity among software developers. As mailing lists were often used to inform the developers about new releases etcetera, new media like Wiki articles [Wik10] (used in the GCC project [Gcc10] for example), blog posts (used in the WordPress project [Wor10] for example) and even Twitter [Twi10] (used in the PHP project [Php10a] for example) messages are used to do this.

Furthermore, discussion boards are being used for having a more structured way of communicating as opposed to mailing lists for example: at discussion boards the messages are arranged by category, topic, etcetera.

### 2.2 Process mining

Process mining is defined as the method of distilling a structured process description from a set of real executions [Aal04, Mar02]. In Figure 2.3, process mining is about the relations between the ‘Model’ and ‘Event logs’ parts: process discovery, conformance and extension. Process discovery uses only the data from the event logs, to derive a process model. Process discovery can for example be used when there are no models available yet. When having a model available, it is possible to check the conformance of that model, to the data in the event logs. Furthermore, it is possible to use the data from the event logs to extend a model, in order to create a model which is a more accurate representation of reality than the original model.

As opposed to data mining, process mining mainly focusses on the temporal relationships between entities. A typical process mining activity consists of extracting a process model (‘which events precede which events’) from a set of events.

Process mining can be classified in three categories: control-flow discovery, performance analysis and organizational analysis [Boz09]. Techniques of each of these categories can be used for process discovery, conformance and extension. Control-flow discovery is concerned with (automatically) constructing a process model describing the causal dependencies between activities [Aal09]. Performance analysis with respect to process mining, includes for example bottleneck analysis (which task/resource causes the maximum throughput time of a case). Organizational analysis is about the relations between originators.

The input data for process mining algorithms, are event logs. Event logs meet the process mining meta model, described in Figure 2.4. Each event log (Log) contains data from a number of processes. In this thesis we assume there is data from only one process. A process contains a number of CaseInstances. Furthermore, a process contains a number of TaskNames. Each case instance contains a number of events.

Figure 2.5 contains a three-dimensional view of a workflow (process model). The three dimensions in this figure are: the resource, process and case dimension. The resource dimension is about the resources involved in the process. A resource (or originator) is for instance an

---

1[http://twitter.com/php_net](http://twitter.com/php_net)
2.2. Process mining

Figure 2.3: Process mining².

![Diagram of Real World and Information System](image)

Figure 2.4: Process mining meta model, modified from [Don05a].

Table 2.1 contains an example of an event log. An event log contains information about the activities performed per case of a workflow.

The Event Type column in Table 2.1 describes the state in the life cycle of the work item.
Workflow are mainly human. However, because workflow management is not restricted to offices, we prefer the term resource. Resources are allowed to deal with specific work items. To facilitate the allocation of work items to resources, resources are grouped into classes. A resource class is a group of resources with similar characteristics. There may be many resources in the same class and a resource may be a member of multiple resource classes. If a resource class is based on the capabilities (i.e. functional requirements) of its members, it is called a role. If the classification is based on the structure of the organization, such a resource class is called an organizational unit (e.g. team, branch or department). A work item which is being executed by a specific resource is called an activity. If we take a photograph of a workflow, we see cases, work items and activities. Work items link cases and tasks. Activities link cases, tasks, and resources.

Figure 2 shows that a workflow has three dimensions: (1) the case dimension, (2) the process dimension and (3) the resource dimension. The case dimension signifies the fact that all cases are handled individually. From the workflow point of view, cases do not directly influence each other. Clearly they influence each other indirectly via the sharing of resources and data. In the process dimension, the workflow process, i.e., the tasks and the routing along these tasks, is specified. In the resource dimension, the resources are grouped into roles and organizational units. We can visualize a workflow as a number of dots in the three dimensional view shown in Figure 2. Each dot represents either a work item (case + task) or an activity (case + task + resource). Figure 2 shows that workflow management is the glue between the cases, the tasks, and the organization.

<table>
<thead>
<tr>
<th>CaseID</th>
<th>Task Name</th>
<th>Event Type</th>
<th>Originator</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>File Fine</td>
<td>Completed</td>
<td>Anne</td>
<td>20-07-2004 14:00:00</td>
</tr>
<tr>
<td>2</td>
<td>File Fine</td>
<td>Completed</td>
<td>Anne</td>
<td>20-07-2004 15:00:00</td>
</tr>
<tr>
<td>1</td>
<td>Send Bill</td>
<td>Completed</td>
<td>system</td>
<td>20-07-2004 15:05:00</td>
</tr>
<tr>
<td>2</td>
<td>Send Bill</td>
<td>Completed</td>
<td>system</td>
<td>20-07-2004 15:07:00</td>
</tr>
<tr>
<td>3</td>
<td>File Fine</td>
<td>Completed</td>
<td>Anne</td>
<td>21-07-2004 10:00:00</td>
</tr>
<tr>
<td>3</td>
<td>Send Bill</td>
<td>Completed</td>
<td>system</td>
<td>21-07-2004 14:00:00</td>
</tr>
<tr>
<td>4</td>
<td>File Fine</td>
<td>Completed</td>
<td>Anne</td>
<td>22-07-2004 11:00:00</td>
</tr>
<tr>
<td>4</td>
<td>Send Bill</td>
<td>Completed</td>
<td>system</td>
<td>22-07-2004 11:10:00</td>
</tr>
<tr>
<td>1</td>
<td>Process Payment</td>
<td>Completed</td>
<td>system</td>
<td>22-07-2004 15:05:00</td>
</tr>
<tr>
<td>1</td>
<td>Close Case</td>
<td>Completed</td>
<td>system</td>
<td>24-07-2004 15:06:00</td>
</tr>
<tr>
<td>2</td>
<td>Send Reminder</td>
<td>Completed</td>
<td>Mary</td>
<td>20-08-2004 10:00:00</td>
</tr>
<tr>
<td>3</td>
<td>Send Reminder</td>
<td>Completed</td>
<td>John</td>
<td>21-08-2004 10:00:00</td>
</tr>
<tr>
<td>2</td>
<td>Process Payment</td>
<td>Completed</td>
<td>system</td>
<td>22-08-2004 09:05:00</td>
</tr>
<tr>
<td>2</td>
<td>Close case</td>
<td>Completed</td>
<td>system</td>
<td>22-08-2004 09:06:00</td>
</tr>
<tr>
<td>4</td>
<td>Send Reminder</td>
<td>Completed</td>
<td>John</td>
<td>22-08-2004 15:10:00</td>
</tr>
<tr>
<td>4</td>
<td>Send Reminder</td>
<td>Completed</td>
<td>Mary</td>
<td>22-08-2004 17:10:00</td>
</tr>
<tr>
<td>4</td>
<td>Process Payment</td>
<td>Completed</td>
<td>system</td>
<td>22-08-2004 14:01:00</td>
</tr>
<tr>
<td>4</td>
<td>Close Case</td>
<td>Completed</td>
<td>system</td>
<td>29-08-2004 17:30:00</td>
</tr>
<tr>
<td>3</td>
<td>Send Reminder</td>
<td>Completed</td>
<td>John</td>
<td>21-09-2004 10:00:00</td>
</tr>
<tr>
<td>3</td>
<td>Send Reminder</td>
<td>Completed</td>
<td>John</td>
<td>21-10-2004 10:00:00</td>
</tr>
<tr>
<td>3</td>
<td>Process Payment</td>
<td>Completed</td>
<td>system</td>
<td>25-10-2004 14:00:00</td>
</tr>
<tr>
<td>3</td>
<td>Close Case</td>
<td>Completed</td>
<td>system</td>
<td>25-10-2004 14:01:00</td>
</tr>
</tbody>
</table>

Table 2.1: An example of an event log [Med08].
2.2. Process mining

Figure 2.6 shows an example of a mined workflow diagram (Petri net), from the event log in Table 2.1. The rectangles correspond to the task names from the table.

![Figure 2.6: A mined workflow (Petri net) from the data in Table 2.1 [Med08].](image)

2.2.1 ProM

In order to apply process mining techniques to event logs, several applications have been developed. An example of such an application is ProM [Pro09].

The ProM process mining workbench is a platform independent, generic open source framework for implementing process mining algorithms in a standard environment [Don05b, Aal07a, Pro09]. Mining, visualization, conversion and other algorithms are available as plugins. ProM contains over 280 plugins.

Figure 2.7 shows an example of the interface of ProM 6. In the middle of this view, the distribution of events per case and event classes (distinct event names) per case are displayed. On the left, statistics about the loaded event log are visible, like the number of cases, events and originators in the log. On the right, information like the start and end dates of the log are displayed. ProM 6 is the current development version. ProM 5 is the latest stable version.

![Figure 2.7: The interface of ProM 6 showing the summary of an event log [Pro09].](image)
Dotted chart analysis

When performing an analysis project of an event log using ProM, the dotted chart analysis [Son07] can be one of the first analysis plugins to use (besides the view presented in Figure 2.7), as the dotted chart provides a high-level overview of the events over time.

The dotted chart is a chart similar to a Gantt chart. The events of the event log are displayed (as dots) over time (horizontally). The events can be categorized (vertically) on several ‘component types’:

- **TaskID**: each distinct event name (like *File Fine*) is displayed in its own ‘lane’.
- **Originator**: each distinct originator (like *Anne*) is displayed in its own ‘lane’.
- **InstanceID**: each case instance (like *1*) is displayed in its own ‘lane’.
- **Event**: each event type (like *Suspended*) is displayed in its own ‘lane’.

Given this categorization, the data can be sorted on several aspects: the *Component Name* (names of the case instances), *Number of Events* (per case instance), *Duration* (time between the first and last event belonging to that case instance), *First Event, Last Event, Actual duration, Actual start time* and *Actual end time*.

The color of the dots can depend on several properties of an event: the *TaskID, Originator, InstanceID* and the *Event*. The vertical lines in the figure subdivide the time line into several periods (days, weeks, months, years, etcetera). This subdivision can be modified using the ‘Time sort (chart)’ parameter.

Figure 2.8 shows an example of the advanced dotted chart plugin of ProM. The advanced dotted chart plugin is very similar to the dotted chart plugin. A difference between them is that when having multiple dots at the same place (given the zoom level), the colors of those events are blended. In that way more information is present about the distribution of events over time. Furthermore, the diameter of the dots can represent the amount of dots at the same place. The more dots, the larger the diameter.

A figure like Figure 2.8 gives information about the arrival pattern of new case instances. Furthermore, it gives information about the amount of events per case instance, as well as the periods in which the number of events per period is high.
2.3 Summary

Software repositories like version control systems, bug trackers and mailing list archives contain historic information about the development process of a software project. Version control systems are used to work on the same piece of information simultaneously by multiple developers, as well as to create backups. Bug trackers are used to store information about bugs found in the software. This information contains the current state of the bug and allows developers to communicate about approaches to solve the bug for example. Mailing list archives contain information about the communication between developers. This communication can consist of announcements, patch distributions and discussions about new features for example.

Process mining consists of methods to extract a structured process description from execution data. Process mining can be used in several ways. When no process model is known, process mining can be used for process discovery. When a process model is known, process mining can be used to test the conformance of that model to the execution data. Furthermore, process mining can be used to extend the available model, in order to get a more accurate model. Process mining can be classified in three categories: control-flow discovery, performance analysis and organizational analysis, focusing on the discovery of process models, bottlenecks and other performance related factors, and the relations between originators, respectively.
Chapter 3

Requirements

As described in the introduction (see Chapter 1), the goal of the project is to create an application which facilitates process analysis of data from various software repositories, in an easy manner. From the project goal, several requirements have been derived for an application. This application has been named ‘FRASR’ (FRamework for Analyzing Software Repositories).

Figure 3.1 describes the scope of FRASR. It uses data from software repositories and processes it in a way that it can be used by analysis applications. The remainder of this chapter describes the requirements of FRASR.

Figure 3.1: The relation of FRASR with data sources and analysis applications.

3.1 Facilitate process analysis

Several applications exist to analyze data from event logs [Pro09, Flu10, Fut10, Pal10]. However, these application are not capable of importing data from software repositories directly.

An application which is capable of importing data from software repositories, is the ProM Import Framework [Pro10]. However, this application only supports logs from SVN and CVS repositories (and other not software repository related data sources). Furthermore, there is only one type of export available (per data source), which cannot be modified (without changing the source of the application). This application is not suited for the project goal.

Therefore, FRASR should provide means to extract the data from the software repositories into a format that can be used by analysis applications. ProM [Pro09] has been chosen as the analysis application for this project, as it is platform independent and it has a large amount of available analysis techniques. The only output format ProM 5 (the latest stable version) supports, is the MXML log format [Don05a]. This format therefore should be supported by FRASR.

Furthermore, as FRASR should support process analysis, the data from the software repositories should be processed to fit the process mining meta model, as presented in Figure 2.4 in
Section 2.2. As the choice of how to link the data significantly influences the possible analysis techniques, this mapping to event log elements should be configurable.

3.2 Various software repositories

When using data from various software repositories, a matching between developers from those sources is necessary since the developer names are one of the few ways to link data from those sources. The names (or aliases) the developers use, can be the same for each software repository, but often this is not the case. Consider for example the usernames from SourceForge and the email addresses developers use at mailing lists.

Besides linking data from software repositories by the aliases of the developers, an other way to link this data, is for instance by finding bug report identifiers in version control system comments. However, these links are not available most of the time [Fis03].

When developer aliases of the various sources are not appropriately matched, the aliases are treated as being separate developers. Treating all the developer aliases as being separate developers may lead to false conclusions about issues like the number of new developers joining a project, and therefore about the cathedral and bazaar phases of a project [Cap07].

For this reason, FRASR should support the matching of developer aliases from the software repositories to a single developer. In this thesis we use ‘developer’ for every human involved in the software development process. These can be for example the users of the software (who have posted a bug report, for example), technical writers (who have changed files in a version control system, for example) or process managers (who have sent email messages, for example).

Furthermore, as new software repositories are being developed, FRASR should be easily extendable to support the data from those repositories.

3.3 Easy manner

The usability of an application can often greatly influence whether the application will be used or not. Therefore, FRASR should implement several requirements related to usability aspects.

3.3.1 Settings

As analysis projects should be reproducible, it has to be possible to save the settings related to the analysis project. These settings include for example those related to the case mapping and the developer matching.

3.3.2 Data processing

Data from software repositories often consists of formatted web pages or version control logs. Processing this data to fit the process mining meta model manually, is infeasible. Having automatic processing of this ‘raw’ data is necessary, since the amount of data is simply too large, and automatic processing is done deterministically. With this requirement, data sources should be identifiable by only providing a URL for example.
3.3.3 Automatic developer matching

The developer matching, as mentioned before, should be included in FRASR to support linking developer aliases from the various data sources. For small projects or projects from which the link between developer aliases is known, the developer matching should be manually constructible in FRASR. However for large projects, it is infeasible to construct this matching manually. Therefore, FRASR should also support automatic developer (alias) matching.

3.3.4 GUI

FRASR should have a Graphical User Interface (GUI) in which all possible tasks can be performed.

3.3.5 Cache

As retrieving data from software repositories may take a considerable amount of time, a cache should be implemented to reduce the loading time of the data from the software repositories, when performing multiple exports of the same data.

3.3.6 Filters

Where a cache can reduce the time to load the data from the software repositories (downloading web pages for example), filters can reduce the time to extract the events and create the event log (given the data from the software repositories). Consider for instance an analysis project which is performed each month. Exporting the data of all the months in the history would be superfluous, as the focus of the current analysis project is only at the last month.

3.4 Summary

From the description of the requirements in the preceding sections, the following list of requirements can be derived:

1. FRASR must perform the (pre)processing of data from the data sources to fit the process mining meta model.
2. The mapping of data from the data sources to the process mining meta model must be configurable.
3. FRASR must have a GUI.
4. It must be possible to save project related settings.
5. FRASR must be able to export the data.
6. Multiple export formats should be available.
7. The MXML format must be supported.
8. FRASR must support the matching of developers aliases between the data sources.
9. It must be possible to apply the developer matching automatically.
10. FRASR should have a cache for the data from the data sources.
11. Filters should be definable which limit the exported data set.

Notice that this list is not very detailed, as basic requirements like ‘FRASR must allow a user to add a new data source’ are not included. However, this list is used as a guidance for the design of FRASR, described in the next chapter.
Chapter 4

FRASR Design

In this chapter we describe the (functional) design of FRASR. The main part consists of the object model (Section 4.1). The developer matching algorithm is discussed in more detail in Section 4.2.

4.1 Object model

Figure 4.1 presents the object model of FRASR. Notice the use of the package symbol as a way to represent hierarchy. The part concerning the data sources is presented in Section 4.1.1 and the part concerning the case mappings is presented in Section 4.1.2. Classes in the object model having a grey background color, represent the points where the model can be extended to add support for new types of data sources, export types, etcetera. Furthermore, only the data fields are visible in the object model. The methods have been left out for readability reasons.

The FRASR class represents an instance of the application. The application has several ExportTypes which can be used to export the data from the data sources (see Requirements 5, 6 and 7 in Section 3.4).

The application has a collection of DataSource and a current Project. A project also has a number of data sources. This separation between the application and the project has been made, to have the developer matching calculated for only the data sources of the project. In this way, information like the number of developers in the project can be extracted from FRASR. When the developer matching would be calculated for all the available data sources, more developers would be extracted, than the number of developers related to the project.

A data source has a CacheOption which determines whether FRASR should attempt at retrieving the data from a (remote) source, a locally stored copy (cache) or both (see Requirements 1 and 10 in Section 3.4). When retrieving data from both sources, the data from the local copy is supplemented with data from the source. When for instance a bug report exists at the local copy, it is not retrieved from the source. When the bug repository contains a bug report which is not at the local copy, it is retrieved from the source. Retrieving all the data from the source to check which version contains the most recent data, would make the cache superfluous. A data source also has a boolean value indicating whether or not data fetched from the source should be added to the cache. The data sources linked to a project can be used in a CaseMapping.

A case mapping defines the way in which data from the data sources are linked to elements of an event log (Figure 2.4). See Section 4.1.2 for more information.
Each project can have a number of Filters attached to it (see Requirement 11 in Section 3.4). The ExportFilters determine which periods in time should be kept in the export and which should be filtered out. The DataSourceFilters determine which events from a data source should be kept in the export, and which events should be filtered out. All the items which do not match the filter, are not included in the export. A data source filter consists of a field (of the linked DataSource), an operator ($>$, $<$, $=$, $\neq$, $\geq$, $\leq$) and a compareValue. With these settings, for instance all the Subversion revisions preceding 435 can be filtered out (data source: Project X, Subversion, field: revision, operator: $\geq$, compareValue: 435).

Each data source can have a number of DeveloperAliases. Each developer alias consists of a number of identifier, name and email values (IdInfo, NameInfo and MailInfo respectively).

An identifier value contains a key-value pair. The key indicates from which set the values are taken. For instance the key is SourceForge and the value is bharat\(^1\). We have chosen this approach to allow different software repositories to have identifier values, which are not from the same set. Consider for example a Subversion repository (where two revisions were committed by the same user, when the usernames of those revisions are the same) and a Bugzilla repository (where two comments on a bug report were posted by the same user, when the email addresses of those comments are the same). Both repositories use identifier fields, but the values are not from the same domain.

A name value contains a value (rawName), Bharat Mediratta for example. A mail value (rawData) contains an email address, possibly combined with a name, B. Mediratta <bharat@example.com> for example. The other fields are derived from this data. The name is B. Mediratta, the address is bharat@example.com and the username

\(^1\)http://sourceforge.net/users/bharat
4.1. Object model

is bharat. More information about the developer matching algorithm is presented in Section 4.2. By means of the developer matching algorithm, or by manual actions, developer aliases can be linked to a Developer. The way in which the link is created is stored in the DeveloperBinding.

4.1.1 Data sources object model

FRASR supports data from various software repositories. Figure 4.2 shows the object model with the data sources currently supported. Notice the DS_<ID> class which represents how FRASR can be extended to support other data sources. See Appendix D.1 for more information on how to extend FRASR with respect to the data source, and Section 9.1 for a list of envisioned extensions. Furthermore, the DataSource class is the point where this figure should be merged with Figure 4.2.

Table 4.1 shows a list of the data sources currently supported by FRASR. These data sources were chosen to have support for a selection of commonly used software repositories.

<table>
<thead>
<tr>
<th>Data source type</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>SourceForge bugs [Sou10]</td>
<td>AWStats [Aws10], Notepad++ [Not10], WinMerge [Win10b]</td>
</tr>
<tr>
<td>DS_SF_Bugs</td>
<td></td>
</tr>
<tr>
<td>SourceForge mails [Sou10]</td>
<td>jEdit [Jed10], PhpMyAdmin [Php10d]</td>
</tr>
<tr>
<td>DS_SF_Mails</td>
<td></td>
</tr>
<tr>
<td>SourceForge forums [Sou10]</td>
<td>AWStats [Aws10], Notepad++ [Not10], PhpMyAdmin [Php10d]</td>
</tr>
<tr>
<td>DS_SF_Forum</td>
<td></td>
</tr>
<tr>
<td>Tigris.org mails [Tig10]</td>
<td>ArgoUML [Arg10], CVSAnalY [Rob04, Cvs10], TortoiseSVN [Tor10]</td>
</tr>
<tr>
<td>DS_TigrisMail</td>
<td></td>
</tr>
<tr>
<td>Mbox mails [Fra08]</td>
<td>Can be used to import data from Outlook [Out10] / Thunderbird [Thu10] for example.</td>
</tr>
<tr>
<td>DS_RawMail</td>
<td></td>
</tr>
<tr>
<td>TRAC wiki [Tra09]</td>
<td>Pidgin [Pid10], ProM [Pro09], WordPress [Wor10]</td>
</tr>
<tr>
<td>DS_TRACWiki</td>
<td></td>
</tr>
<tr>
<td>TRAC tickets [Tra09]</td>
<td>FileZilla [Fil10], Gallery [Gal10], WordPress [Wor10]</td>
</tr>
<tr>
<td>DS_TRACTickets</td>
<td></td>
</tr>
<tr>
<td>Subversion [Sub09]</td>
<td>Apache [Apa10], ArgoUML [Arg10], jEdit [Jed10]</td>
</tr>
<tr>
<td>DS_SVN</td>
<td></td>
</tr>
<tr>
<td>Bugzilla [Bug09]</td>
<td>ArgoUML [Arg10], GCC [Gcc10], KDE [Kde10]</td>
</tr>
<tr>
<td>DS_Bugzilla</td>
<td></td>
</tr>
<tr>
<td>JIRA bugs [Jir10]</td>
<td>Apache [Apa10], Hibernate [Hib10], Spring Framework [Spr10]</td>
</tr>
<tr>
<td>DS_JIRA_Bugs</td>
<td></td>
</tr>
<tr>
<td>MARC mail [Mar10a]</td>
<td>Apache [Apa10], PHP [Php10a], Linux [Lin10]</td>
</tr>
<tr>
<td>DS_MARCMail</td>
<td></td>
</tr>
<tr>
<td>Piper mail [Mai10]</td>
<td>phpList [Php10c], Weka Miner [Wek10], WordPress [Wor10]</td>
</tr>
<tr>
<td>DS_PiperMail</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: The data sources currently supported by FRASR.
4.1.2 Case mappings object model

When doing process mining, the way in which the cases are chosen, greatly influences the possible analysis techniques which are applicable to the data from the event log. Therefore, there are several methods in FRASR, in which the data from the data sources can be mapped onto a case (see Requirement 2 in Section 3.4).

Figure 4.3 presents the object model of the case mapping part from Figure 4.1. As with the previous figures, the classes with a (light) grey background color indicate the places in which FRASR can be extended. See Appendix D for more information on how to extend FRASR. The classes which have a grey line and text color represent classes from the ‘high level’ model. Notice also that the methods have been left out for readability reasons.

A Case mapping contains a number of EventBindings. The type of the case mapping determines the way the case instances are derived from the events of the data source of the event binding. The type of the event binding determines what the event name of the event will be. See Section 4.1.3 for more information about the model used for the case instances and events etcetera.

Case mappings

FRASR has five different types of case mappings. All the case mappings except the ConstantCase derive the case instances (for the events) from the data of the events.

Constant case The constant case allows an analyst to define the CaseInstances manually. Defining case instances manually can be useful when the case instance of an event is not relevant (for instance when analyzing the distribution of tasks per developer).

Developer case The DeveloperCase uses the originator of an event (the developer) as the case instance. Using the developer case mapping, the way developers work can be compared.
Datetime case  As programmers are humans, they sometimes make changes which cause problems. Śliwerski et al. presented a technique to find fix-inducing changes; the changes which caused the problems [Śli05]. In their work, they found that for the MOZILLA project [Moz10], the most fix-inducing changes were made on Fridays. As of future work, they state that this information can be combined with other properties of a change (a revision). To analyze for instance the developer group further, the datetime case can be used in FRASR. The datetime case has several variants, listed in Table 4.2. Each variant takes the time stamp of an event, and uses it as an input for the case instance. Notice the added 0 in the Month of the year variant for example, which is useful for sorting purposes.

Field case  Some data sources have fields which identify artifacts. The FieldCase can use these fields as the case instance for the events. In a Bugzilla [Bug09] repository for example, the bug_id field is used to identify a bug report. When using the values of this field as the case instance, an analyst can get insight into the lifecycle of a bug report: ‘are bug reports often re-assigned’, ‘are bug reports closed in batches or regularly in time’, etcetera. Notice that the data fields which are selected (for each linked event binding) are not included in Figure 4.3 for readability reasons.

Component case  Software typically consists of several components or packages etcetera. Sometimes it is not possible to test all the components with the same ‘attention’ (for example due to time or cost constraints). In that case, the testing should be focussed on components which are likely to contain bugs. A method to compare components, is to look at the developer
<table>
<thead>
<tr>
<th>Variant</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day of the week</td>
<td>Wednesday</td>
</tr>
<tr>
<td>Day of the month</td>
<td>24</td>
</tr>
<tr>
<td>Day of the year</td>
<td>83</td>
</tr>
<tr>
<td>Week of the year</td>
<td>12</td>
</tr>
<tr>
<td>Month of the year</td>
<td>03</td>
</tr>
<tr>
<td>Year-day</td>
<td>2010-83</td>
</tr>
<tr>
<td>Year-week</td>
<td>2010-12</td>
</tr>
<tr>
<td>Year-month</td>
<td>2010-03</td>
</tr>
<tr>
<td>Year</td>
<td>2010</td>
</tr>
</tbody>
</table>

Table 4.2: Variants of the datetime case in FRASR. All the output examples are generated from the input: 2010-03-24 15:16:17.

activity. When the activity of some components is low in the last period, but high for other components, one might want to focus the testing on the ‘high activity components’ [Nag07]. To analyze the activity of the components of a system, the ComponentCase can be used. For each linked event binding, the component case records the way in which the components are extracted from the data. There are three variations to link an event binding: Path, Field and Text.

- The path variant has a list of regular expressions as an input. The first match replaces the path of a file (/trunk/src/com/component/componentX.java) by the replace value (componentX). This variant is useful for data from a version control system for example.
- The field variant takes a field of the data source as the component name. To ‘normalize’ the names of the component names, it is possible to provide a list of regular expressions (compX can be replaced by componentX for example). Like in the path variant, only the first match will be applied. This variant can for instance be used for data from a bug tracker, which often has a special field for the component of the source code the bug is related to.
- The text variant takes a list of keywords as an input, and calculates the frequencies of the keywords in a list of fields from the data source. The keyword with the highest frequency is used as the name for the component. Like in the field variant, using regular expressions, this name can be ‘normalized’. This variant can be used for data from a mailing list for example.

Notice that the information about the selected variations per linked event binding is not incorporated in the model in Figure 4.3 for readability reasons.

**Event bindings**

FRASR contains a variety of event bindings (or bindings). Event bindings are used to link a data source to a case mapping. An event binding defines which events are extracted from the data in the data source.

**Basic event bindings** Event bindings which are not specific to a data source are the ConstantEventBinding, DataSourceNameEventBinding, DataSourceTypeEventBinding and
4.1. Object model

**DataSourceCategoryEventBinding.** Each of those bindings, retrieves all the elements from the data source (for a Subversion [Sub09] repository, these are the revisions and the modifications (belonging to the revisions)) and creates an event for it. For all those extracted events, the event name is derived from a property of the data source.

- **ConstantEventBinding:** The event name for each event is CONST. This binding can be used when the name of the event is irrelevant for the analysis question (for example what the number of events per developer is).
- **DataSourceNameEventBinding:** The event name for each event is the name of the data source, as provided by the user. This name can be for example: Project X, Subversion 3. This binding can be used to distinguish between events from two Subversion repositories for example.
- **DataSourceTypeEventBinding:** The event name for each event is the type of the data source. For Project X, Subversion 3 the type can be: Subversion. This binding can be used to distinguish between events from two types of bug trackers, for example a JIRA bugs database and a Bugzilla database.
- **DataSourceCategoryEventBinding:** The event name for each event is the category of the data source. For Project X, Subversion 3 the category can be: Version control system. This binding can be used to distinguish between events from two types of systems, for example a Subversion repository and a Bugzilla database.

**Detailed event binding** An other event binding which is not specific to any data source type, is the detailed event binding. Using this binding, it is possible to specify for every node in the hierarchy of the data fields of the data source, whether to use it, and what the event names should be. Consider the following example of a Bugzilla database:

- **Bugzilla database**
  - L(Reports)
    - L(Comments)
      - Comment information
    - L(Updates)
      - Update information
      - Report information

  The Bugzilla database consists of a list (L) of bug reports. Each bug report contains information (a number of fields like bug.id, owner, etcetera) and a list of comments and a list of updates. Comments are messages posted by users and updates contain information about how and when the fields of the bug report were modified.

  The detailed event binding allows to indicate for each level (so reports, comments and updates in this example) whether to use the events from this level, and which field should be used as the event name. The available fields for a level, are the fields of that level and the fields of all the ‘higher’ levels. So for the updates, also the fields of the bug report are available. Besides the fields, also other options are available to set the event name to the options from the basic event bindings (constant, data source name, data source type and data source category).

  Using the detailed event binding in combination with the field case for example, it is possible to get for each file in a Subversion repository (used as a case instance) the modifications which were performed on it (as events).
Data source event bindings Besides the event bindings which are applicable to every type of data source, there are also special event bindings which are only applicable to a single data source type. These special event bindings use the data source characteristics to generate the events. The following events are generated by the bindings:

- **SourceForge bugs**: Ticket-created, Ticket-commented, Ticket-closed and Ticket-reopened.
- **SourceForge mails, MARC mail, Mbox mails, Tigris.org mails and Piper mail**: Mail thread created, Mail reply and Mail latest-reply. A thread with 20 messages for example, has 1 Mail thread created event, 18 Mail reply events and 1 Mail latest-reply event.
- **SourceForge forums**: Forum topic created, Topic reply and Topic latest-reply. A topic with 5 messages for example, has 1 Forum topic created event, 3 Topic reply events and 1 Topic latest-reply event.
- **TRAC wiki**: Wiki article created, Wiki article change and Wiki article latest-change. A thread with 11 messages for example, has 1 Wiki article created event, 9 Wiki article change events and 1 Wiki article latest-change event.
- **TRAC tickets**: Ticket-created, Ticket-updated, Ticket-assign, Ticket-reopen, Ticket-accepted, Ticket-closed(\(<\text{resolution}\>)\) where \(<\text{resolution}\>)\) can be: du- plicate, worksforme, invalid, wontfix
- **Subversion**: VCS: A, VCS: M, VCS: D, VCS: R. For each revision, the most frequent modification type of the modified files/directories are calculated. VCS: A stands for addition, VCS: M for modification, VCS: D for deletion and VCS: R for rename.
- **Bugzilla**: Ticket-created, Ticket-commented, Ticket-updated and Ticket-\(<\text{status}>[\!(\(<\text{resolution}>\))]\) where \(<\text{status}>\) and \(<\text{resolution}>\) can be values from Figure 2.2 at page 5. Notice that \(<\text{resolution}>\) is optional.
- **JIRA bugs**: Bug-created, Bug-updated, Bug-resolved and Bug-commented

The data sources which create Mail latest-reply events, create these events only for the message of the thread with the latest date. Messages belong to the same thread, when their subject (without “Re:” prefixes) is equal.

Data source component event bindings An other type of data source type specific event bindings are the data source component event bindings. These event bindings are similar to the component case variants, but then the derived component names are used as the event names instead of the case instance.

4.1.3 Log object model

Figure 4.4 shows the internal log model used in FRASR. Such a model is necessary to allow easy extendability of the application, for other export types. Notice that the figure does not show the variables and methods.

Furthermore, notice that for using the AttributeModel, the case mapping object model as described in Section 4.1.2 should be extended to also define mappings from the data elements to log attributes.
4.2 Developer matching

Analyzing data from a software project with multiple software repositories, requires integration of developer aliases. Without such integration, all the aliases would appear as single developers. In that way, no data is linked between the software repositories. Notice that it also can be the case that developers use several aliases in one software repository (for example because credentials were lost).

In small projects the matching of developer aliases can be constructed manually, but for large projects with thousands of developer aliases (for instance in Apache [Apa10], ArgoUML [Arg10] or Mozilla [Moz10]), it is infeasible to match those aliases manually.

FRASR can automatically calculate a developer matching for the developer aliases of the data sources linked to a project (see Requirements 8 and 9 in Section 3.4).

4.2.1 Algorithm

Algorithm 1 describes the matching of a list of developer aliases. The object model (see Figure 4.1) shows that a developer can have 0 or more developer aliases attached, and a developer alias can be attached to 0 or 1 developer. Notice that when a developer alias is not attached to a developer, but does appear in the data source (and the export), a default developer is used (and created when necessary).

Algorithm 1 uses a hierarchical clustering technique [Man08] to calculate the matching. The first step in the developer matching algorithm is to create singletons (link every developer alias to a separate developer, lines 2-4). Next, for every pair of developer aliases, the match value (distance function, indicating how much the given developer aliases match) is calculated (lines 5-11). When this value is higher than a threshold value (which can be set by the user), the developers of the given developer aliases are merged. See Chapter 6 for more information about the effect of the weights to the amount of aliases that will be matched and the relation between the threshold value and the percentage of matched developer aliases. Notice that Algorithm 1 is quadratic in the number of aliases.

We have chosen to use an approach which uses a threshold value in combination with weights, to allow the user to influence the percentage of matched aliases without having
to manually adjust the automatically constructed matching. Especially for projects with thousands of developer aliases.

The merging process links all the aliases of the developer of one of the aliases, to the developer of the other alias. Figure 4.5 presents an example in which all the aliases of Developer 2 are linked to Developer 1, given that there was a match between Alias $\alpha$ and Alias $\epsilon$.

<table>
<thead>
<tr>
<th>Developer 1 - Alias $\alpha$</th>
<th>Developer 1 - Alias $\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Alias $\beta$</td>
<td>Merge</td>
</tr>
<tr>
<td>- Alias $\gamma$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>Developer 2 - Alias $\delta$</td>
<td>- Alias $\delta$</td>
</tr>
<tr>
<td>- Alias $\epsilon$</td>
<td>- Alias $\epsilon$</td>
</tr>
</tbody>
</table>

Figure 4.5: An example of the merging process at the developer matching.

Algorithm 1: Developer matching.

Algorithm 2 describes the calculation of the match value between two developer aliases. The first step is to calculate the match objects for each developer alias (lines 3-4). There are two types of match objects: IdMatchObjects and TextMatchObjects. An IdMatchObject contains a key and a value, and a TextMatchObject contains only a value. See the description of Algorithm 3 for more information. The match objects from both developer aliases are pairwise compared. When the types match (line 5), a comparison is registered. When the match objects match (line 7), the product of the weights is added to the matchSum. For IdMatchObjects, the key and the value have to be equal and for TextMatchObjects, the value has to be equal.

The match value between the developer aliases is calculated as the division of matchSum by numComparisons. Notice that as the weights of a match object are reals ranging from 0 to 1 (included), the calculated match value is also a real value ranging from 0 to 1 (included).

Using this approach, pairs of developer aliases with a higher number of matching fields have a higher match value than pairs of developer aliases with a lower number of matching
4.2. Developer matching

Furthermore, fields which are more likely to indicate a correct match (identifier fields for example), can have more influence on the matching process.

| Input          | Two DeveloperAlias objects: DeveloperAlias1 and DeveloperAlias2.          |
| Output         | A real ranging from 0 to 1 (included) where a higher number indicates a better match. |

1. numComparisons ← 0;
2. matchSum ← 0;
3. foreach mo1 ∈ matchObjects(DeveloperAlias1) do
   4. foreach mo2 ∈ matchObjects(DeveloperAlias2) do
      5. if type(mo1) = type(mo2) then
         6. numComparisons ← numComparisons + 1;
         7. if hasMatch(mo1, mo2) then
            8. matchSum ← matchSum + mo1.weight * mo2.weight;
         end
      end
   end
3. end
4. return 0 when numComparisons = 0, else matchSum / numComparisons

Algorithm 2: Developer matching, calculates the matchValue between two developer aliases.

Algorithm 3 describes the creation of match objects for a developer alias. A developer alias has a number (0 or more) of IdInfo, NameInfo and MailInfo fields. Each of those fields is converted to a number of match objects. There are two types of match objects: IdMatchObject and TextMatchObject. Each of those types has a weight field (fullIdWeight, rawNameWeight and mailUsernameWeight for example; all reals ranging from 0 to 1 (included)) and a value (a string, which is converted to lower case characters and leading and trailing whitespace omitted) field. See Chapter 6 for more information about the weights. An IdMatchObject also has a key field (formatted in the same way as the value field).

IdInfo fields are converted to an IdMatchObject and a TextMatchObject (lines 2-5). The TextMatchObject is added to compare the value to other name/mail values.

NameInfo fields are converted into two TextMatchObjects (lines 6-8). One for the ‘raw’ name, which is the name including special characters/punctuation etcetera, and one for the parsed version of the raw name, not including special characters/punctuation etcetera. Furthermore, when the parsed name contains 2 words, combinations of the first name and the first character of the last name (and vice versa) are added as TextMatchObjects (lines 9-17). These combinations are only included when their length is at least minCombinationLength. Similar approaches are described in [Bir06, Rob05].

MailInfo fields, like NameInfo fields, are converted into a number of TextMatchObjects (lines 19-21). First a match object is created for the mail address. Next, a match object is created for the username part of the address (the part before the symbol). Then when the mail info contains a name associated with the mail address, there is a match object created for the combination of the name and address (lines 22-25). Furthermore, match objects are created for the name, in the same way as match objects have been created for the other NameInfo fields.
**Input**: A DeveloperAlias object.

**Output**: A list of match objects.

1. $l ←$ empty list of match objects;
2. foreach $id ∈ IdInfo$ do
   3. add IdMatchObject($id.key, id.value, fullIdWeight$) to $l$;
   4. add TextMatchObject($id.value, valueldWeight$) to $l$;
3. endforeach
4. foreach $name ∈ NameInfo$ do
   5. add TextMatchObject($name.raw, rawNameWeight$) to $l$;
   6. add TextMatchObject($name.parsed, parsedNameWeight$) to $l$;
   7. if $name.parsed$ contains 2 words then
      8. // $f$ is the first part of $name.parsed$ and $s$ is the second part
      9. // Only add the objects when:
         10. length of the concatenation $\geq \text{minCombinationLength}$
         11. // First takes the first character of a string
         12. add TextMatchObject($f + \text{First}(s), \text{combinedNameWeight}$) to $l$;
         13. add TextMatchObject($\text{First}(s) + f, \text{combinedNameWeight}$) to $l$;
         14. add TextMatchObject($s + \text{First}(f), \text{combinedNameWeight}$) to $l$;
         15. add TextMatchObject($\text{First}(f) + s, \text{combinedNameWeight}$) to $l$;
   16. end
17. endforeach
18. foreach $mail ∈ MailInfo$ do
19. add TextMatchObject($mail.address, fullMailWeight$) to $l$;
20. add TextMatchObject($mail.username, mailUsernameWeight$) to $l$;
21. if $mail$ has name attached then
22. add TextMatchObject($mail.raw, fullMailInformationWeight$) to $l$;
23. add name of $mail$ to $l$ like the previous $name ∈ \text{NameInfo}$;
24. end
25. endforeach
26. return $l$

**Algorithm 3**: Developer matching, creating the objects to match.

### 4.2.2 Properties

The developer matching algorithms uses match objects of a developer alias to calculate the match with another developer alias. The number of match objects, the number of matching objects and the weights influence the resulting $\text{matchValue}$ of two developer aliases. Equation 4.1 describes the calculation of the $\text{matchValue}$ between two developer aliases.

\[
\text{matchValue} = \frac{\sum_{i=1}^{\text{numMatches}} (w_{i1} \cdot w_{i2})}{\text{numComparisons}}
\]  

(4.1)

where $\text{numComparisons}$ is the number of compared match objects, $\text{numMatches}$ is the number of matching match objects, and $w_{i1}$ and $w_{i2}$ are the weights of a match object from developer alias 1 and 2, respectively.
4.2.3 Examples

Table 4.3 contains the default developer matching settings of FRASR. Higher values indicate a higher importance for the weights. Setting the threshold to a low value (like 0.1), results in a higher number of matched developer aliases than setting it to a high value (like 0.8). Setting the threshold to a value ‘close’ to 0, results in matching two aliases when there exists at least one match between the match objects of those developers.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullIdWeight</td>
<td>1</td>
</tr>
<tr>
<td>valueIdWeight</td>
<td>0.8</td>
</tr>
<tr>
<td>rawNameWeight</td>
<td>0.8</td>
</tr>
<tr>
<td>parsedNameWeight</td>
<td>0.6</td>
</tr>
<tr>
<td>combinedNameWeight</td>
<td>0.3</td>
</tr>
<tr>
<td>fullMailInformationWeight</td>
<td>0.8</td>
</tr>
<tr>
<td>fullMailWeight</td>
<td>0.6</td>
</tr>
<tr>
<td>mailUsernameWeight</td>
<td>0.6</td>
</tr>
<tr>
<td>threshold</td>
<td>0.001</td>
</tr>
<tr>
<td>minCombinationLength</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.3: Default developer matching settings of FRASR.

Using the settings from Table 4.3, we present a number of examples of the developer matching algorithm. The first example compares two identical IdInfo fields, and the second example compares an IdInfo field with a MailInfo field.

Example 1: One of the developers from the WinMerge project [Win10b] at SourceForge [Sou10] is Christian List. His username at SourceForge is christianlist. In a number of data sources of the WinMerge project (see Appendix A.7 for more information), the developer alias christianlist is present. This alias consists of one IdInfo field having christianlist as the value and SourceForgeUserId as the key. Notice that users at SourceForge also have a User ID, but as the username is also unique, it can be used as the key. Table 4.4 shows the details of both aliases. Using Algorithm 2, the calculated matchValue is \((1 * 1 + 0.8 * 0.8)/2 = 0.82\). Notice that for a matchValue of 1, the valueIdWeight should be 1.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Aliased field 1</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias 1</td>
<td>christianlist</td>
<td></td>
</tr>
<tr>
<td>Alias 2</td>
<td>christianlist</td>
<td></td>
</tr>
<tr>
<td>IdInfo</td>
<td>SourceForgeUserId</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>christianlist</td>
<td>0.8</td>
</tr>
<tr>
<td>MatchObject</td>
<td>IdMatchObject</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TextMatchObject</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: Details of the first developer matching example.

\[\text{http://sourceforge.net/users/christianlist}\]
Example 2: Consider two fictional developer aliases, as described in Table 4.5. Alias 1 is an email address and Alias 2 is a username from a Subversion repository. Using Algorithm 2, the calculated matchValue is \((0.6 \times 0.8)/2 = 0.24\). Notice that only the TextMatchObjects generate comparisons, as there is no IdMatchObject for Alias 1.

<table>
<thead>
<tr>
<th>Alias</th>
<th>Alias 1</th>
<th>Alias 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias name</td>
<td><a href="mailto:psmith@example.com">psmith@example.com</a></td>
<td>psmith</td>
</tr>
<tr>
<td>MailInfo</td>
<td><a href="mailto:psmith@example.com">psmith@example.com</a></td>
<td>-</td>
</tr>
<tr>
<td>IdInfo</td>
<td>-</td>
<td><a href="http://example.com/svn/psmith">http://example.com/svn/psmith</a></td>
</tr>
<tr>
<td>MatchObject</td>
<td>TextMatchObject</td>
<td>IdMatchObject</td>
</tr>
<tr>
<td></td>
<td>TextMatchObject</td>
<td>TextMatchObject</td>
</tr>
<tr>
<td>Weight</td>
<td>0.6</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 4.5: Details of the second developer matching example.
Chapter 5
FRASR Implementation

This chapter describes the implementation of FRASR. FRASR has been implemented using Java, as Java is a platform independent language. The settings of the data sources in the application and the project settings are stored in XML files. This format allows the data to be easily imported at another machine for example.

The implementation of the cache and the GUI are described in Section 5.1 and Section 5.2 respectively. The 3rd party software libraries which have been used are discussed in Section 5.3.

5.1 Cache

The current implementation of the cache of FRASR, stores the downloaded data directly in files. So when for instance a web page containing a bug report is retrieved, this entire page is stored on the disk. This approach allows to manually copy the entire cache to another machine for example, without having to perform any other actions. When using a MySQL database for example, the data first has to be exported, and then imported again, keeping in mind that certain tables have to exist etcetera.

The cache stores the data for each data source in a separate folder. Each data element (for example a bug report) is stored in a separate file. In order to retrieve all bug reports from a data source, all the files in the associated directory have to be listed. The current implementation of the cache does not check whether a bug report from the source is newer than the bug report from the cache. Retrieving all the data from the source to check which version contains the most recent data, would make the cache superfluous. For some data sources, where the date of the last modification of a data element (for example a bug report) is known, other strategies can be implemented using this information, to further decrease the loading time when using the cache.

The size of the cache is limited by the amount of disk space of the machine FRASR runs on. Table 5.1 shows the size of the cache for a number of SourceForge projects. As most machines nowadays contain several gigabytes of storage, the size of the cache is not likely to reach this limit.

A number of experiments have been conducted to validate whether the cache reduces the time to load the data\textsuperscript{1}. For a number of SourceForge projects, first the developer aliases have been loaded from the source only (and stored to the cache). Next, the developer aliases

\textsuperscript{1}All the experiments have been performed on a 32bit Windows7 machine with an Intel Core2 Quad CPU @ 2.40 GHz with 3GB of memory, using the TU/e internet connection.
were reloaded, but now from the cache only. Loading the developer aliases for a project, ensures that all the data is loaded from the data sources. Table 5.1 shows the results of the experiments. For all the projects, the use of the cache significantly reduces the time to load all the developer aliases. Notice that the Subversion logs were not cached, as the used 3rd party library does not support this, leaving room for a further decrease in loading times. Furthermore, only the Subversion repositories were used, as FRASR currently has no support for other version control systems. For the same reason, the only topics included are those from the forums at SourceForge.

<table>
<thead>
<tr>
<th>Project</th>
<th>#revisions</th>
<th>#bugs</th>
<th>#mails</th>
<th>#topics</th>
<th>Cache size</th>
<th>Source time</th>
<th>Cache time</th>
</tr>
</thead>
<tbody>
<tr>
<td>aMSN</td>
<td>12062</td>
<td>3137</td>
<td>34947</td>
<td>-</td>
<td>222 MB</td>
<td>75 min</td>
<td>10 min</td>
</tr>
<tr>
<td>Gallery</td>
<td>20976</td>
<td>6774</td>
<td>73825</td>
<td>-</td>
<td>543 MB</td>
<td>198 min</td>
<td>39 min</td>
</tr>
<tr>
<td>jEdit</td>
<td>18178</td>
<td>6750</td>
<td>88220</td>
<td>-</td>
<td>542 MB</td>
<td>208 min</td>
<td>39 min</td>
</tr>
<tr>
<td>Notepad++</td>
<td>639</td>
<td>4763</td>
<td>33143</td>
<td>11818</td>
<td>358 MB</td>
<td>436 min</td>
<td>17 min</td>
</tr>
<tr>
<td>PhpMyAdmin</td>
<td>13465</td>
<td>7694</td>
<td>49696</td>
<td>10065</td>
<td>568 MB</td>
<td>588 min</td>
<td>45 min</td>
</tr>
<tr>
<td>ScummVM</td>
<td>50733</td>
<td>6948</td>
<td>88220</td>
<td>11818</td>
<td>619 MB</td>
<td>266 min</td>
<td>52 min</td>
</tr>
<tr>
<td>WinMerge</td>
<td>7223</td>
<td>3349</td>
<td>4256</td>
<td>-</td>
<td>143 MB</td>
<td>71 min</td>
<td>10 min</td>
</tr>
</tbody>
</table>

Table 5.1: Decrease in time exporting data from SourceForge projects, using the cache.

5.2 GUI

As FRASR should be easy to use, a GUI has been implemented in which all the necessary actions can be performed to export data from software repositories to a format which is supported by the chosen analysis application (see Requirement 3 in Section 3.4). The GUI has been implemented using Java Swing Components. These components include various standard components for forms and file handling windows etcetera.

The GUI of FRASR allows the user to perform all the actions necessary to define the data sources, manage the cache, calculate the developer matching for a project, save the project settings (see Requirement 4 in Section 3.4), etcetera. Appendix B describes the GUI in further detail.

5.3 External libraries

FRASR uses a number of external libraries to extract and export data from the data sources, as well as to save and load XML setting files:

- SVNKit [Svn10], used for the Subversion data source.
  - Version: 1.3.2.6267
  - License: The TMate Open Source License permits you to use SVNKit at no charge under the condition that if you use the software in an application you redistribute, the complete source code for your application must be available and freely redistributable under reasonable conditions.

- JavaMail [Jav10b], used for the Mbox mail and Piper mail data sources.
  - Version: 1.4.3
5.3. External libraries

- **Licence**: CDDL-1.0, BSD, GPL-2.0

- SuperCSV [Sup10], used for the Bugzilla and Trac tickets data sources.
  - **Version**: 1.52
  - **Licence**: Apache Software License

- TagSoup [Tag10], used for the SourceForge mails, SourceForge bugs, SourceForge forums, Bugzilla, Tigris mail, Trac wiki, MARC mail and Piper mail data sources.
  - **Version**: 1.2
  - **Licence**: Apache License v2.0

- XML files [Jdo10], used to manage the settings files of FRASR.
  - **Version**: 1.1
  - **Licence**: Apache-style open source license, with the acknowledgment clause removed

- MXML [Don05a, Pro10], used for the MXML export of FRASR.
  - **Version**: 4.0
  - **Licence**: BSD
Chapter 6

Developer Matching Experiments

The threshold value influences the outcome of the developer matching algorithm. Figure 6.1 shows the results of experiments concerning the recall of the developer matching algorithm for a number of small projects for which the optimal matching is known. These projects are TU/e software engineering projects, which consist of about 7-9 students per group. The optimal matching has been constructed using information from the project documents. See Section 7.1 for more information about those projects. Three specific software engineering projects (Prifes, PRINCESS and Octosep) have been chosen, as these were the only projects where data was available from at least a Subversion repository and a mailing list archive.

For all the experiments, the default developer matching settings have been used (as presented in Section 4.2.3). Figure 6.1 shows a decrease in recall for increasing threshold values. The precision of all the experiments was 1, indicating that there were no false positives for these experiments.

One of the causes for the decrease in recall in Figure 6.1 is relatively low weight (0.3) for the name combination match objects. These match objects are useful for matching usernames from the Subversion repositories to the username parts from an email addresses. The repositories from the software engineering projects contain aliases which have data from Subversion repositories and mailing list archives. Therefore, these aliases can be matched by name combination match objects. Consider for example the name John Smith and the email address jsmith@example.com. The name is ‘combined’ to jsmith, which is matched with the username part of the email address (jsmith). As these match objects have a relatively low weight, their share in the calculation of the match value is also relatively small. When using a higher threshold value, this share is not sufficient to have a match, resulting in a decrease in recall. Furthermore, for the Prifes project, the name combination matches caused the decrease in recall for threshold value 0.24.

For the three software engineering projects, the recall is less than 1 for a threshold value of 0. The reason for this is that aliases from ‘system’ users, like trac for example, are not matched. Furthermore, aliases which have names with multiple insertions (like Jan van de Velde) are not matched with email addresses (like j.v.d.velde@example.com) correctly.

Figure 6.1 shows a number of points (threshold values) where the recall decreases. These points are related to the ‘types’ of developer aliases. A number of developer aliases only

---

1Recall is a measure of completeness. In this setting, recall is the fraction of matched aliases which are present in the optimal matching, compared to the optimal matching. See [Man08] for more information.

2Precision is a measure of exactness. In this setting, precision is the fraction of matched aliases which are also in the optimal matching, compared to the matched aliases. See [Man08] for more information.
Figure 6.1: Decreasing recall for increasing threshold value.

Figure 6.2 shows the result of seven of those projects. Per project, the developer matching has been calculated with a threshold value between 0 and 1 (including), with steps of 0.01. For every calculation, the default weights have been used. Appendix A contains detailed information about the used data sources, the number of aliases extracted from those data sources, etcetera.

As expected, Figure 6.2 shows a decreasing trend: a higher threshold value gives a lower number of matched developer aliases. Furthermore, for a threshold value of 0, the match percentage varies from 35% until 54%. This ‘initial’ percentage is related to the number of developers having only one alias attached. Having only one alias attached, implies that there is no match with other developer aliases. The more developers with only one developer alias present in a developer matching, the lower the match percentage.
Figure 6.2: Decreasing match percentage for increasing threshold value.

Figure 6.2 shows for each project a number of horizontal line segments: equal match percentage for decreasing threshold values. This is also present in Figure 6.1. Notice that it is easier to manually split the aliases of a developer into two clusters (developers), than to manually find two related clusters (and merge them). Therefore, for TextMatchObjects, the matching can be improved by using the Levenshtein edit distance for example, to check whether two objects match. This technique has been used by Bird et al. for instance [Bir06]. Notice that this technique should not be used for the IdMatchObjects, as identifiers should be unique.
Chapter 7
Case Studies

To demonstrate the applicability of FRASR for applying process mining techniques to data from multiple software repositories, two case studies have been conducted. The first case study has been conducted to demonstrate the applicability of process mining techniques to data from software repositories. The second case study has been conducted to demonstrate the added value of using multiple repositories.

Figure 7.1 describes the typical use of FRASR for answering a Software Engineering question (S.E. Question). Given this question, FRASR is used to process the data from the available software repositories to an event log. This event log is then analyzed in an external application. The application chosen for the case studies, is the ProM process mining workbench, as it is a platform independent application, which has a large amount of available analysis process mining techniques (called plugins). With the analysis, the initial software engineering question can be answered.

The use of FRASR can be summarized in four steps:

- **Step 1:** Define the data sources: provide information about the data sources (like a URL and authentication information).
- **Step 2:** Define the case mapping: the case mapping defines in which way the data from the data sources is processed to the event log.
- **Step 3:** Calculate the developer matching: this step is necessary to match the data from the various data sources to developers.
- **Step 4:** Export: export the data to an event log which can be used by the analysis application.
Detailed information about how the case studies were carried out (detailed settings of the case mappings etcetera) can be found in Appendix C.

For all the software repositories used in the case studies, we assume that the time stamps are synchronous, meaning that when time stamps from both repositories are equal, the points in ‘real time’ they were recorded, do not differ significantly.

7.1 Case study 1: TU/e software engineering project

The goal of this case study is on the one hand to prove the applicability of process mining techniques to data from software repositories, and on the other hand to show that FRASR can be used to quickly get insights which otherwise would require interviews with the involved developers.

For this case study, the TU/e software engineering projects [Sep10] have been chosen, as these projects have a fixed size (in terms of resources, deadlines, development process). Furthermore, detailed information about the projects is known, which can be used to validate the results obtained in this case study.

Each computer science bachelor student at the Eindhoven University of Technology has to participate (in the 3rd year) in a project with 7-9 other students. The project aims at familiarization of the students with working in a large non-trivial software project. The project has to be carried out by the ESA software engineering standards [ESA91, Jon97]. These standards include a number of guidelines which the students must adhere to, like the prescribed software development model, the use of the prototype, the artifacts to be created, etcetera. The people involved in a software engineering project are the students, the project manager (a master student), the technical advisor (typically a member of the academic staff) and the senior management (members from the academic staff).

The senior management has formulated a number of guidelines. From these guidelines, the following analysis questions have been formulated:

- Question 1: How strictly has the V-model been applied?
- Question 2: Did the project manager perform his tasks?
- Question 3: Was the prototype a part of the final implementation?

Questions 1 and 3 are posed from the perspective of the project manager and the senior management, and Question 2 is posed from the perspective of the senior management.

**Question 1** A recent study about the TU/e software engineering projects indicates that the currently used software development model could be changed to better fit the characteristics of the projects [Koe09]. In order to get insight in the execution of the currently used development model, the only feedback available at the moment is data from questionnaires and discussions with group members and project managers. As this information might be incorrect or incomplete, it is interesting to verify this information using the data available in the software repositories.

**Question 2** Currently, the senior management only has the feedback from the group members as an indicator of the performance of the project manager. In order to verify this information, the data from the available software repositories can be used. In that way the senior management can quickly get insight into the performance of the project manager.
Question 3 With respect to prototypes, the ESA software engineering standards [ESA91, Jon97] state the following:

Prototypes usually implement high risk functional, performance or user interface requirements and usually ignore quality, reliability, maintainability and safety requirements. Prototype software is therefore ‘pre-operational’ and should never be delivered as part of an operational system.

As with Question 2, the senior management can only ask the project members whether they respected this rule. In order to check the information provided by the group members, data from the version control system of the project can be used.

7.1.1 Question 1: How strictly has the V-model been applied?

The Prifes TU/e software engineering project was carried out in 2008, using the V-model [Moo01]. In this model there are several phases, each with a corresponding test phase, see Figure 7.2\(^1\).

The use of this model requires each phase to be completed before starting work on the next phase. Furthermore, for each phase the corresponding test plan also has to be completed in that phase. Hence, to test how strictly the V-model has been applied, one can study to what extent the phases overlap, and whether the test plans were created at the right point in time.

Step 1: Define the data sources

For this analysis, all the available data sources from the Prifes project have been used. The available data sources for this project are a Subversion repository, Trac tickets, Trac wiki and a mailing list archive. Appendix C.1 describes the details of all the data sources.

\(^1\)http://www.onestoptesting.com/images/v-model.jpg
Step 2: Define the case mapping

The available data sources have been combined, having the artifact as a case instance (using the component case, as described in Section 4.1.2).

For data from the Subversion repository, for every revision an event is created. The case instance of these events is the first match of the path name of the modified files, with a given list of regular expressions. A similar approach has been used by Rubin [Rub07]. The approach by Rubin only applies the described technique to derive the event names. Furthermore, the approach by Rubin does not define the order in which the regular expressions are verified. When having input which matches multiple regular expressions, the order determines which regular expression is applied. The approach used by FRASR assumes that all files belonging to a revision are from the same artifact. A manual inspection of the event log confirmed that this assumption holds. This assumption allows us to use the revisions as an event, for the Subversion repository. Using the revisions (instead of the modified files), provides a more ‘readable’ figure in the advanced dotted chart, as the number of very large dots is smaller (indicating a large number of modified files at the same point in time, given the current zoom level). When having an event for each modified file, updating all the code (changing the copyright notice in each file for example), would appear as if there was a lot of work performed on the code, as it only represents one activity, namely ‘updating the copyright notice’.

For data from the Trac tickets, the case instances for the events are derived from the ‘component’ field, which directly maps onto the artifacts. The events from the mailing list and the Trac wiki are mapped onto the artifacts, by using the artifact with the highest frequency in the messages body and subject of the associated messages. This approach assumes that the keyword with the highest frequency is most likely to be the best representation of the content of the message.

The events for the mailing list, the Trac tickets and the Trac wiki are created by their associated data source bindings, as described on page 24. See Appendix C.1.1 for more details.

Step 3: Calculate the developer matching

The developer matching has been constructed manually, as I was personally involved in this project. This resulted in a perfect developer matching.

Step 4: Export

The defined case mapping from Step 2 has been exported to an MXML event log. Figure 7.3 shows the advanced dotted chart analysis of the exported log. The horizontal axis describes the time. The vertical axis describes the artifacts, sorted by the time stamp of the first event of that artifact. The orange vertical lines indicate the deadlines of the phases of the project as established by the senior management:

- User requirements phase: 2008-02-25.

\(^2\)Modified in this setting can also be added, deleted or renamed.
7.1. Case study 1: TU/e software engineering project

- Implementation or detailed design (DD) phase (internal deadline): 2008-05-19.

The other vertical lines indicate the weeks. The artifacts are (from top to bottom): **URD** (User Requirements Document), **Code**, **SRD** (Software Requirements Document), **ATP** (Acceptance Test Plan), **ADD** (Architectural Design Document), **STP** (System Test Plan), **ITP** (Integration Test Plan), **DDD** (Detailed Design Document) and **UTP** (Unit Test Plan). Other artifacts have been filtered out.

Artifacts belonging to the same project phase, have the same color. **URD** and **ATP** for the user requirements phase, **SRD** and **STP** for the software requirements phase, **ADD** and **ITP** for the architectural design phase, **Code**, **DDD** and **UTP** for the implementation phase.

![Figure 7.3: Overlap between project phases of the Prifes project.](image)

When looking at the overlap between the phases, one can see that all the phases overlap with the adjacent phases. Furthermore, throughout the entire project, work has been done considering the code (see Section 7.1.3 for a more detailed analysis). The overlap does not show that the main artifacts (**URD**, **SRD**, **ADD** and **DDD**) were developed simultaneously. The overlap between all the adjacent phases was a result of the fact that the deadlines were quite close to each other. This resulted in completing the artifacts in time to be approved by the customer and the technical advisor. Shifting the beginning of the next phase back in time gained extra time to work on the artifacts³.

Not all the test plans were developed according to the V-model. The **ATP** (Acceptance Test Plan) for instance, should have been completed in the user requirements phase, but development on this artifact took place during the entire project and starting only at the end of the user requirements phase³.

An other observation which can be made regarding Figure 7.3 is the fact that there has been ‘work’ concerning the **URD** after the SR phase. The first event (at 27-03-2008) is a modification of a Wiki article containing an overview of all the minutes of the project (a minute from a meeting in the UR phase was not yet present at this page). This event was classified as being part of the UR phase, as the modification contained information related to the UR phase. The second event (at 21-04-2008) is a modification of a Wiki article containing all the

³This information was not extracted from the software repositories, but was known as I was personally involved in this project
information regarding meetings with the customer (which had not been updated since halfway the SR phase). This event was classified as being part of the UR phase, as the modification contained information related to the UR phase. The work on the SRD and ADD consisted of tickets assigned to group members and modifications to the documents, as a consequence of a change requests for the SRD and the ADD, regarding consistency between the implementation and the design documents. The events regarding the SRD and the ADD between AT1 and AT2 are related to preparing the documents (copying to a special folder) for handing them over to the customer and senior management. The event regarding the Code after AT2 is the creation (VCS: A) of the final version of the source code (tagging).

Table 7.1 presents the results obtained for other software engineering projects. For most of the projects, there has been overlap between at least two phases. The results were confirmed by the project managers, group members and the senior management.

The study by Koenraadt [Koe09] states that the senior management wishes to experiment more with an iterative model. The data of future projects using this model can be analyzed by using FRASR, to compare the results to the results of this case study.

<table>
<thead>
<tr>
<th>Project</th>
<th>Year</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dempsey</td>
<td>2005</td>
<td>There was some overlap between the UR and SR phase, and the AD and DD phase. There was no significant overlap between the UR and AD phase. As all the test plans were incorporated in the SVVP (Software Verification and Validation Plan), no information about the separate test plans is available.</td>
</tr>
<tr>
<td>DAVIS</td>
<td>2006</td>
<td>There was little overlap between the phases.</td>
</tr>
<tr>
<td>Diffusion</td>
<td>2007</td>
<td>There was little overlap between the UR and SR phase and the AD and DD phase, but quite some overlap (about the length of half a phase) between the SR and AD phase.</td>
</tr>
<tr>
<td>PRINCESS</td>
<td>2009</td>
<td>There was little overlap between the UR and SR phase, but quite some overlap between the SR and AD phase and the AD and DD phase.</td>
</tr>
<tr>
<td>Octosep</td>
<td>2009</td>
<td>Regularly during the SR phase, work has been performed on artifacts from the UR phase. There is no significant overlap between the SR and AD phase, as well as between the AD and DD phase.</td>
</tr>
</tbody>
</table>

Table 7.1: Results for other software engineering projects, showing similar results.
7.1.2 Question 2: Did the project manager perform his tasks?

The tasks of a project manager in a TU/e software engineering project include:

- Chairing the meetings (and distributing the agenda for each meeting).
- Creating and updating the Software Project Management Plan (SPMP).

Steps 1, 2 & 3: Define the data sources, define the case mapping & calculate the developer matching

To analyze the way in which these tasks were performed, the same event log as for Question 1 has been used, but it has been filtered only to include the events from the project manager. The name of the project manager has been extracted from the SPMP.

Step 4: Export

Figure 7.4 shows the events regarding the Agendas and the SPMP, with which the project manager has been involved. The vertical lines (orange) indicate the deadlines of the project, as described in the previous section.

![Figure 7.4: The events of the project manager regarding the agendas and the SPMP, per project phase.](image)

With respect to the distribution of the events from the Agenda case instance, the agendas were regularly send. Every week (except for holiday/examination weeks), there has been a message to the mailing list by the project manager, about the agenda. A manual verification using the dates from the agendas available at the Subversion repository confirmed that indeed, the agendas were send for each meeting. This verification was necessary as the messages from the project manager could also have been messages informing the project group that there was no agenda for the weekly meeting.

The second part of the analysis concerns creating and updating the SPMP. Figure 7.4 shows at which points in time the SPMP has been modified by the project manager. Notice that the user requirements phase started at 2008-02-01. In this period, there have been no events by the project manager related to the SPMP, which should have been the case. By using the unfiltered version of the event log (as used for Question 1), we observed that one of the project members created the SPMP in the user requirements phase.

Table 7.2 presents the results for other software engineering projects. These results show significant difference between the project managers of the various projects. Notice that the total number of agendas per project differs, as this is related to the total duration of the project (which changed from two trimesters to a semester) and the number of exam weeks and holidays.

---

4The Software Project Management Plan contains information about the organization of the project (people involved, process model, responsibilities, etcetera), the technical process (tools, documentation, etcetera), the work packages, the schedule, etcetera.
The results were confirmed by the project managers, group members and the senior management. The senior management indicated that they had the feeling that some of the project managers did not perform their tasks very well.

<table>
<thead>
<tr>
<th>Project</th>
<th>Year</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dempsey</td>
<td>2005</td>
<td>The project manager added 11 out of 18 agendas to the Subversion repository. The project manager did not create the SPMP. The project manager did not modify the SPMP in the AD phase.</td>
</tr>
<tr>
<td>DAVIS</td>
<td>2006</td>
<td>The project manager only had three events in the Subversion repository, of which one considered the SPMP and one considered an agenda. The entire repository only has three events related to an agenda. A possible explanation is that a mailing list has been used (of which the data was not available). The Subversion repository only has three events related to the SPMP.</td>
</tr>
<tr>
<td>Diffusion</td>
<td>2007</td>
<td>The project manager added 15 out of 17 agendas to the Subversion repository. The project manager created the SPMP, and updated it before every ‘next’ phase.</td>
</tr>
<tr>
<td>PRINCESS</td>
<td>2009</td>
<td>The project manager communicated all the 14 agendas to the group members. Furthermore, the SPMP was created and every phase updated by the project manager.</td>
</tr>
<tr>
<td>Octosep</td>
<td>2009</td>
<td>For only 4 out of 12 agendas, events related to the project manager have been found in the data sources. The project manager had no events related to the SPMP. In the data sources, only 4 events were registered as being related to the SPMP.</td>
</tr>
</tbody>
</table>

Table 7.2: Results for other software engineering projects, related to the performance of the project manager.

7.1.3 Question 3: Was the prototype a part of the final implementation?

For this analysis question, we focus on the source code of two projects: Diffusion (2007) and Prifes (2008).

Step 1: Define the data sources

For each project, the Subversion repository is used to create the event log.

Step 2: Define the case mapping

The files (and folders) contained in the Subversion repository are used as the case instances, and the actions performed on them (addition, modification, etcetera) are used as the event names.

Step 3: Calculate the developer matching

The same developer matching has been used as for Question 1 and Question 2.
Step 4: Export

The exported event logs are filtered (in ProM) to keep only the source files. See Appendix C.1.3 for more information about the settings which were used to export and filter the event logs.

Prifes  
Figure 7.5 shows the advanced dotted chart analysis of the filtered event log from Prifes. The horizontal axis describes the time and the vertical axis describes the code files and folders, ordered by the time stamp of the first event. The colors indicate the type of the action (blue for addition, white for modification, black for rename and red for deletion). The red triangles have been manually added.

Each of those triangles indicates a period in which related files have been modified (or added, deleted or renamed). The first triangle ranges from the beginning of the project until halfway the project and the second triangle ranges from halfway the project until the end of the project. As the events in the first triangle end close to the deadline of the SR phase, there is an indication that these files belong to the prototype, which should have been developed during the SR phase [ESA91]. Furthermore, there is an indication that the prototype files were not reused during the actual development, as the prototype files were not modified after the SR deadline.

Notice that the files were not copied to another folder, as that would have resulted in a large vertical blue line (the line around 2008-05-20 is the creation of a branch). It is possible that the contents of the files was used in the final implementation, this requires further analysis, by inspecting the contents of the files.

Figure 7.5: The Prifes source files used for the prototype and the final implementation.
Diffusion  Figure 7.6 shows a similar chart as Figure 7.5, obtained for the Diffusion project. In this figure, there is only one triangle present, starting at the beginning of the project and ending at the end of the project. Having only one such triangle present could indicate that the prototype has been (partly) used in the final implementation.

The presence of one triangle can also indicate that there was no prototype (or not in the form of source code, but ‘on paper’ for example). However, in that case the triangle should ‘start’ from about halfway the project, corresponding to the beginning of the AD phase.

The other software engineering projects investigated (Dempsey, DAVIS, PRICESS & Octosep) showed similar results as the Prifes project. Therefore 5 out 6 analyzed projects met the requirement that the prototype may not be (a part) of the final implementation. These results were confirmed by the group members and project managers. The senior management indicated that regulations regarding the prototype are not verified at the moment, as these are considered as a less relevant.

7.1.4 Conclusion

The results of this case study indicate that in every analyzed project, there is overlap between the phases in the V development model. In only one of the six analyzed projects, the project manager communicated all the agendas to the project members. In two of the six analyzed projects, the project manager created the SPMP and updated it before every next phase. In only one of the analyzed projects, the prototype might have been used as a part of the final implementation.

These results have been confirmed by the involved group members, project managers and
the senior management. Especially the results regarding the project managers were regarded as interesting by the senior management, as they had the feeling some of the project managers were not performing their tasks very well.

From these results, it follows that applying process mining techniques to data from software repositories can provide insights which could not have been obtained by existing techniques. Furthermore, FRASR can be used to get insights in the development process of software projects, without having to communicate to the people involved.

Notice however, that the results obtained in this case study do not always present hard evidence. For instance the absence of agendas sent over a mailing list does not have to mean that the agendas were not delivered to the project members, as these could have been handed out on paper. In those cases further assessment may be necessary.

7.1.5 Monitoring

Using FRASR, a software engineering project can be analyzed. The analysis questions in this chapter have all been answered using data from completed software engineering projects. Not all possible analysis questions require the project to be completed. For example the distribution of the agendas by the project manager can also be analyzed halfway the project. This allows FRASR to be used for monitoring purposes.
Chapter 7. Case Studies

7.2 Case study 2: Open source software development

The goal of this case study is to demonstrate the added value of using the combination of multiple software repositories compared to using multiple data sources separately. We show that when using the ‘combined’ data from multiple software repositories, we can get insights which cannot be achieved by using only data from the single software repositories in isolation.

Software projects can be classified in two categories: closed source and open source. Open source software development is regarded as a successful model of “natural product evolution” [Nak02]. Therefore, it is interesting to analyze the development process of open source software projects. Analyzing these projects gives insights into properties of this development process, which can be used to improve the development process of other projects, and to compare different open source projects with each other [Cro03].

One of the aspects of the open source software development model, is the organizational structure. Besides the evolution of the system, the evolution of the development community can be studied to get insights into the open source software development model. Nakakoji et al. propose a classification of the development process of an open source into three types: Exploration-Oriented, Utility-Oriented, and Service-Oriented. A classification of the development process of open source software projects can provide guidance on the creation and maintenance of sustainable open source software development and communities [Nak02].

A part of the analysis of the organizational structure of an open source project, is the classification of developers to roles [Nak02]. Such a classification is necessary to analyze the social relationships between the community members and the relationships between the roles.

The classification defined by Nakakoji et al. has been chosen as it contains a detailed description of the roles, as opposed to other classifications like the one given by Crowston and Howison [CH05] for example. The classification as defined by Nakakoji et al. is:

- **Passive user.** Passive Users just use the system in the same way as most of us use commercial software; they are attracted to OSS (Open Source Software) mainly due to its high quality and the potential of being changed when needed.

- **Reader.** Readers are active users of the system; they not only use the system, but also try to understand how the system works by reading the source code. Readers are like peer reviewers in traditional software development organizations.

- **Bug reporter.** Bug Reporters discover and report bugs; they do not fix the bugs themselves, and they may not read source code either. They assume the same role as testers of the traditional software development model.

- **Bug fixer.** Bug Fixers fix the bug that is either discovered by themselves or reported by Bug Reporters. Bug Fixers have to read and understand a small portion of the source code of the system where the bug occurs.

- **Peripheral developer.** Peripheral Developers contribute occasionally new functionality or features to the existing system. Their contribution is irregular, and the period of involvement is short and sporadic.

- **Active developer.** Active Developers regularly contribute new features and fix bugs; they are one of the major development forces of OSS systems.

- **Core member.** Core Members are responsible for guiding and coordinating the development of an OSS project. Core Members are those people who have been involved with the project for a relative long time and have made significant contributions to the
development and evolution of the system. In some OSS communities, they are also called Maintainers.

- **Project leader.** Project Leader is often the person who has initiated the project. He or she is responsible for the vision and overall direction of the project.

Nakakoji *et al.* furthermore state:

_Not all of the eight types of roles exist in all OSS communities, and the percentage of each type varies. Each OSS community may have different names for the above roles._

The remainder of this section describes the analysis project, which uses the classification.

### 7.2.1 Project: aMSN

The chosen software project for this analysis project, is aMSN [Ams10]. aMSN is free and open source Windows Live Messenger [Win10a] clone. Windows Live Messenger is a free instant messaging application, created by Microsoft. Besides the Windows platform (used for Windows Live Messenger), aMSN also supports other platforms like Macintosh, UNIX/Linux etcetera. The aMSN application has been downloaded more than 20 million times. aMSN has been chosen, since a list\(^5\) of developers is available, created by the developers involved. This list can be used to validate the outcome of the analysis.

Two of the categories from the list, presented in Section 7.2, cannot be used for this analysis project, as there is no usable data available for those categories. The category ‘passive user’ is not considered for this analysis project, since there is only information available about the number of times the executables were downloaded, but there is no registration of which users were involved. Furthermore, as the source files of all SourceForge projects are available without registration, it is also not possible to distinguish users with the ‘reader’ role. Therefore, we only consider the categories: bug reporter, bug fixer, peripheral developer, active developer, core member and project leader.

#### Step 1: Define the data sources

The data sources used for the analysis of the aMSN project are 7 bug repositories, 3 email archives and 1 Subversion repository. Appendix A.1 contains the settings used for these data sources. The data ranges from February 26, 2002 until July 09, 2010. The aMSN project also has a discussion forum. The data of this forums cannot be used in the current implementation of FRASR.

#### Step 2: Define the case mapping

The data is exported using the developer case, and the data source specific binding for each data source.

#### Step 3: Calculate the developer matching

Appendix A.1 describes the number of aliases extracted per data source of the aMSN project. The developer matching has been calculated using the default settings, as described in Chapter 6.

\(^5\) [http://www.amsn-project.net/current-developers.php](http://www.amsn-project.net/current-developers.php)
Step 4: Export

The defined case mapping from Step 2 has been exported to an MXML event log. Using the exported data, we use the following criteria to classify a developer in one of these categories:

- **Project leader:**
  Involved from the beginning of the project\(^6\), having \textit{VCS: A} and \textit{VCS: M} events.

- **Core member:**
  Developers which have been involved in the project for a relatively long period of time. We have chosen 3 years or more of continuous (at least one event every month) activity. The project started in 2002, resulting in 8 years of development at the moment. We chose 3 years as this is more than a quarter of the entire development time. The developers furthermore must have made significant contributions. We defined significant in this setting to have more revisions in the version control system than average (of the developers having events in the version control system). The developers must have \textit{VCS: A} and \textit{VCS: M} events.

- **Active developer:**
  Developers regularly (for a shorter period of time than core members) having \textit{Ticket-closed}, \textit{VCS: A} and \textit{VCS: M} events. The period from the first event until the last event (of a developer) should be continuous (at least one event every month).

- **Peripheral developer:**
  Developers having \textit{VCS: A} and \textit{VCS: M} events, occasionally (there is no restriction on having an event every month).

- **Bug fixer:**
  Developers having \textit{Ticket-closed} events, and \textit{VCS: M} events, and not belonging to the other categories.

- **Bug reporter:**
  Developers having \textit{Ticket-created} events or \textit{Mail thread created}\(^7\), but no \textit{VCS: X} (where \(X = A, M, R\) or \(D\)) events.

In this definition, the categories are assigned in reverse order, so first the project leaders roles are assigned, next the core member roles, etcetera.

Notice that these criteria can also be defined on a specific period of the project, for example one year. Analyzing all the years of which the project have been under development provides insight in the evolution of the development community. For this case study, we focus on the entire development process.

Using the previously defined criteria, the developers extracted by the developer matching algorithm of FRASR were assigned one of the roles.

Figure 7.7 gives an overview of the developer activity during the 8 years of development (aMSN started in 2002). The vertical axis of this advanced dotted chart [Son07], describes the developers, sorted ascending by the time stamp of the first event of that developer.

\(^{6}\)Nakakoji \textit{et al.} state that the project manager is often involved from the beginning of the project. The other criteria (responsible for vision and overall direction of the project) are hard to formalize. Therefore, we require a project leader to be involved from the beginning of the project. Furthermore, the given classification at the website of the project does not state at which date it was last updated. Notice that this analysis project can also be focussed on a specific period.

\(^{7}\)Samples of the mailing list messages have been inspected, confirming that the mailing list is often used for reporting bugs.
7.2. Case study 2: Open source software development

The horizontal axis describes the time line. The vertical lines indicate the years. The dots represent the events of the developers. The dots are colored by type: Mail thread created, Mail reply and Mail latest-reply are green. Ticket-created is black. Ticket-closed, Ticket-commented and Ticket-reopened are red. VCS: A is blue. VCS: M, VCS: D and VCS: R are white.

Figure 7.7: Developer activity of the aMSN project.

Notice the developer at the top of Figure 7.7 (Developer: -null/-null-). This developer contains all the data which was posted anonymously. Table 7.3 shows the details of this developer, as extracted by the 'originator by task matrix' plugin in ProM [Med08]. The columns under the Mail heading represent the number Mail thread created, Mail reply and Mail latest-reply events. The columns under the Ticket heading represent the number of Ticket-created, Ticket-commented, Ticket-closed and Ticket-reopened events. The columns under the VCS heading represent the number of VCS: A, VCS: M, VCS: D and VCS: R events. Total represents the total number of events for that user.

Figure 7.8 presents the advanced dotted chart of the developers of the aMSN project using the same settings as used for Figure 7.7, only sorting the developers descending by the number of events and zooming in on the top 15 developers. Using this view (in ProM on all the developers, without zooming), and the ‘originator by task matrix’ plugin in ProM in combination with a spreadsheet application, the project leaders, core members, active developers

---

8 The name of a developer is derived from the associated aliases. In this case, there were two aliases (one having only an IdInfo field, and one having only a NameInfo field), which were combined using a separator (/).

9 This matrix contains for all originators (developers), per event, the number of times that event was executed by that developer.
Table 7.3: Data related to developer: -null-/null-.

<table>
<thead>
<tr>
<th>Developer</th>
<th>Mail</th>
<th>Ticket</th>
<th>VCS</th>
<th>Total</th>
</tr>
</thead>
</table>

and peripheral developers were extracted.

First the the advanced dotted chart has been used to classify the project leaders (being involved from the beginning of the project). For each of those developers, the spreadsheet application has been used to check whether they have the required events. Next the core members were extracted using the advanced dotted chart (checking whether the developers have a continuous development period of at least 3 years), and they were checked with the spreadsheet application for having the required events. Similarly the active developers and peripheral developers were extracted.

![Figure 7.8: Developer activity of the aMSN project, sorted by the number of events (top 15).](http://www.amsn-project.net/current-developers.php)

For the bug fixer role, we also used the ‘originator by task matrix’ in combination with a spreadsheet application. The developers having Ticket-closed and VCS: M events, and which were not already assigned to a role were assigned the bug fixer role.

Using the ‘originator by task matrix’ in combination with a spreadsheet application, we filtered out the developers without a role assignment, and having Ticket-created or Mail thread created events, without having VCS: A, VCS: M, VCS: D or VCS: R events. These developers were assigned the bug reporter role.

Table 7.4 presents the number of developers assigned to each role. The Other category consists of developers having only Ticket-commented or Mail reply or Mail latest-reply events (6 max). In the Other category, there furthermore is one developer (SourceForge Robot/sf-robot), which has 128 Ticket-commented and 128 Ticket-closed events.

Tables 7.5 until 7.10 present the developers assigned to each role. For all the developers presented by the project\(^\text{10}\), the associated developer extracted by FRASR has been linked (manually).

\(^\text{10}\)http://www.amsn-project.net/current-developers.php
7.2. Case study 2: Open source software development

<table>
<thead>
<tr>
<th>Role</th>
<th># developers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug reporter</td>
<td>1443</td>
</tr>
<tr>
<td>Bug fixer</td>
<td>3</td>
</tr>
<tr>
<td>Peripheral developer</td>
<td>29</td>
</tr>
<tr>
<td>Active developer</td>
<td>6</td>
</tr>
<tr>
<td>Core member</td>
<td>7</td>
</tr>
<tr>
<td>Project leader</td>
<td>3</td>
</tr>
<tr>
<td>Other</td>
<td>234</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1725</strong></td>
</tr>
</tbody>
</table>

Table 7.4: Developers assigned to each role.

**Role: Bug reporter**

Table 7.5 presents the top 10 of developers having the bug reporter role assigned, sorted by the number of Ticket-created and Mail thread created events.

<table>
<thead>
<tr>
<th>Developer</th>
<th>Mail</th>
<th>Ticket</th>
<th>VCS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rafael/rafaelandy</td>
<td>38</td>
<td>2</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>Kenneth Hedlund/zoddym</td>
<td>0</td>
<td>1</td>
<td>26</td>
<td>45</td>
</tr>
<tr>
<td>Luis manson/psychoboy</td>
<td>23</td>
<td>0</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>Kemal Ilgar Eroglu/ilgar</td>
<td>4</td>
<td>1</td>
<td>13</td>
<td>26</td>
</tr>
<tr>
<td>Carlos Alberto Morales Ramirez/c4x</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>Juan Martin Lopez/nauj27</td>
<td>13</td>
<td>0</td>
<td>1</td>
<td>17</td>
</tr>
<tr>
<td>Sherif Zaroubi/szaroubi</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>arpi49/arpi49</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Javier Ortega Conde/malkavian666</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>GrdScarabe/grdscarabe</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 7.5: Developers classified as bug reporter (top 10).

**Role: Bug fixer**

Table 7.6 presents the developers having the bug fixer role assigned. The data is sorted by the number of Ticket-closed events.

<table>
<thead>
<tr>
<th>Developer</th>
<th>Mail</th>
<th>Ticket</th>
<th>VCS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luis Q. T./elezeta</td>
<td>3</td>
<td>15</td>
<td>1</td>
<td>78</td>
</tr>
<tr>
<td>jonne/jonne_z</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>93</td>
</tr>
<tr>
<td>Alberto Diaz/yozko</td>
<td>4</td>
<td>17</td>
<td>2</td>
<td>92</td>
</tr>
</tbody>
</table>

Table 7.6: Developers classified as bug fixer.

**Role: Peripheral developer**

Table 7.7 presents the top 10 of developers having the peripheral developer role assigned. The data is sorted by the number of VCS: A events.

<table>
<thead>
<tr>
<th>Developer</th>
<th>Mail</th>
<th>Ticket</th>
<th>VCS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luis Q. T./elezeta</td>
<td>3</td>
<td>15</td>
<td>1</td>
<td>78</td>
</tr>
<tr>
<td>jonne/jonne_z</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>93</td>
</tr>
<tr>
<td>Alberto Diaz/yozko</td>
<td>4</td>
<td>17</td>
<td>2</td>
<td>92</td>
</tr>
</tbody>
</table>
Table 7.7: Developers classified as peripheral developer (top 10).

Role: Active developer

Table 7.8 presents the developers having the active developer role assigned. The data is sorted by the total number of events.

Table 7.8: Developers classified as active developer.

Role: Core member

Table 7.9 presents the developers having the core member role assigned. The data is sorted by the total number of events.

Table 7.9: Developers classified as core member.
7.2. Case study 2: Open source software development

Role: Project leader

Table 7.10 presents the developers having the project leader role assigned. The data is sorted by the total number of events.

<table>
<thead>
<tr>
<th>Developer</th>
<th>Mail C</th>
<th>Mail R</th>
<th>Mail L</th>
<th>Ticket Cr</th>
<th>Ticket Co</th>
<th>Ticket Cl</th>
<th>Ticket RO</th>
<th>VCS A</th>
<th>VCS M</th>
<th>VCS D</th>
<th>VCS R</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alvaro J. Iradier Muro/airadier</td>
<td>337</td>
<td>1</td>
<td>11</td>
<td>6</td>
<td>591</td>
<td>635</td>
<td>2</td>
<td>116</td>
<td>2291</td>
<td>34</td>
<td>0</td>
<td>4024</td>
</tr>
<tr>
<td>Philippe Khalaf/burgerman</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>72</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>147</td>
<td>0</td>
<td>0</td>
<td>229</td>
</tr>
<tr>
<td>D. Emilio Grimaldo T./lordofscripts</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>18</td>
<td>10</td>
<td>0</td>
<td>14</td>
<td>108</td>
<td>0</td>
<td>0</td>
<td>152</td>
</tr>
</tbody>
</table>

Table 7.10: Developers classified as project leader.

Table 7.11 presents the developers as defined on the aMSN project website\textsuperscript{11}, compared to the developers extracted from the software repositories of the aMSN project. Project name represents the name of the developer as defined on the project website. \textit{(Role)} represents the role as described on the project page. The list of abbreviations can be found in Table 7.12. FRASR name represents the name as extracted by FRASR. \textit{Role} represents the classified role (R for bug reporter, F for bug fixer, P for peripheral developer, A for active developer, C for core member and L for project leader). The columns under the \textit{Mail} heading represent the number \textit{Mail thread created}, \textit{Mail reply} and \textit{Mail latest-reply} events. The columns under the \textit{Ticket} heading represent the number of \textit{Ticket-created}, \textit{Ticket-commented}, \textit{Ticket-closed} and \textit{Ticket-reopened} events. The columns under the VCS heading represent the number of VCS: A, VCS: M, VCS: D and VCS: R events. Total represents the total number of events for that user.

Table 7.11 does not contain a matching developer extracted by FRASR, for every developer from the project website. A reason for this is that it could be that only a nickname was used for the software repositories, making it impossible to match the aliases. An other reason can be that it is a special case, like Dave Mifsud, which is indicated as the original author of the predecessor of the aMSN project. However, for 35 out of the 41 developers presented at the website a ‘match’ has been found. For all the developers having the A (Admin) role, the analysis classified 4 out of 6 of them as a core developer, one as a project leader and one as a bug fixer. The developer Arieh Schneier, classified as an active developer, mostly has VCS: M, \textit{Ticket-created} and \textit{Ticket-closed} events. This matches the role Win Maintainer, as a maintainer fixes bugs, creates and closes tickets. The same holds for Jerome Gagnon-Voyer, which has the \textit{Maintainer} role and is classified as a core developer. Developer Khalaf G. Philippe is classified as a project leader, however, he is not indicated as one of the developers who started the project.

\textsuperscript{11}http://www.amsn-project.net/current-developers.php
<table>
<thead>
<tr>
<th>People helped along the way</th>
<th>Start</th>
<th>Current</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aitor Ruiz de Samaniego</td>
<td>R</td>
<td>29</td>
<td>38</td>
</tr>
<tr>
<td>Rafael Rodriguez</td>
<td>R</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Tom Hennigan/tomhennigan</td>
<td>T</td>
<td>281</td>
<td>337</td>
</tr>
<tr>
<td>Thomas Geirhovd/tg90nor</td>
<td>T</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>Jonne Zutt</td>
<td>R</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Cristofaro Del Prete</td>
<td>R</td>
<td>18</td>
<td>22</td>
</tr>
<tr>
<td>Karol Krizka/kkrizka</td>
<td>L</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol J. Iradier</td>
<td>L</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Alvaro J. Iradier Muro/airadier</td>
<td>L</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Alaoui Youness/kakaroto</td>
<td>A,D,P</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Vivia Nikolaidou</td>
<td>C</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Sander Hoentjen/tjikkun</td>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Philippe Vandenbrouck</td>
<td>P</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>Boris Faure/billiob</td>
<td>B</td>
<td>1041</td>
<td>1041</td>
</tr>
<tr>
<td>Alexander Nestorov</td>
<td>N</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Monaghan</td>
<td>A,D</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Didimo Emilio Grimaldo T.</td>
<td>D</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Philippe Monaghan - Philip</td>
<td>D</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Malo</td>
<td>G</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Danilo Troccoli</td>
<td>G</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>Karel Demeyer/scapor</td>
<td>D</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Tom Jenkins</td>
<td>T</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Aitor Ruiz de Samaniego</td>
<td>R</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Rafael Rodriguez</td>
<td>R</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Jonne Zutt</td>
<td>R</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Cristofaro Del Prete</td>
<td>R</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol Krizka/kkrizka</td>
<td>L</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol J. Iradier</td>
<td>L</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Alvaro J. Iradier Muro/airadier</td>
<td>L</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Alaoui Youness/kakaroto</td>
<td>A,D,P</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Vivia Nikolaidou</td>
<td>C</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Sander Hoentjen/tjikkun</td>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Philippe Vandenbrouck</td>
<td>P</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>Boris Faure/billiob</td>
<td>B</td>
<td>1041</td>
<td>1041</td>
</tr>
<tr>
<td>Alexander Nestorov</td>
<td>N</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Monaghan</td>
<td>A,D</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Philippe Monaghan - Philip</td>
<td>D</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Malo</td>
<td>G</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Danilo Troccoli</td>
<td>G</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>Karel Demeyer/scapor</td>
<td>D</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Tom Jenkins</td>
<td>T</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Aitor Ruiz de Samaniego</td>
<td>R</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Rafael Rodriguez</td>
<td>R</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Jonne Zutt</td>
<td>R</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Cristofaro Del Prete</td>
<td>R</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol Krizka/kkrizka</td>
<td>L</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol J. Iradier</td>
<td>L</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Alvaro J. Iradier Muro/airadier</td>
<td>L</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Alaoui Youness/kakaroto</td>
<td>A,D,P</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Vivia Nikolaidou</td>
<td>C</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Sander Hoentjen/tjikkun</td>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Philippe Vandenbrouck</td>
<td>P</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>Boris Faure/billiob</td>
<td>B</td>
<td>1041</td>
<td>1041</td>
</tr>
<tr>
<td>Alexander Nestorov</td>
<td>N</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Monaghan</td>
<td>A,D</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Philippe Monaghan - Philip</td>
<td>D</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Malo</td>
<td>G</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Danilo Troccoli</td>
<td>G</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>Karel Demeyer/scapor</td>
<td>D</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Tom Jenkins</td>
<td>T</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Aitor Ruiz de Samaniego</td>
<td>R</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Rafael Rodriguez</td>
<td>R</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Jonne Zutt</td>
<td>R</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Cristofaro Del Prete</td>
<td>R</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol Krizka/kkrizka</td>
<td>L</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol J. Iradier</td>
<td>L</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Alvaro J. Iradier Muro/airadier</td>
<td>L</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Alaoui Youness/kakaroto</td>
<td>A,D,P</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Vivia Nikolaidou</td>
<td>C</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Sander Hoentjen/tjikkun</td>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Philippe Vandenbrouck</td>
<td>P</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>Boris Faure/billiob</td>
<td>B</td>
<td>1041</td>
<td>1041</td>
</tr>
<tr>
<td>Alexander Nestorov</td>
<td>N</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Monaghan</td>
<td>A,D</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Philippe Monaghan - Philip</td>
<td>D</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Malo</td>
<td>G</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Danilo Troccoli</td>
<td>G</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>Karel Demeyer/scapor</td>
<td>D</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Tom Jenkins</td>
<td>T</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Aitor Ruiz de Samaniego</td>
<td>R</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Rafael Rodriguez</td>
<td>R</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Jonne Zutt</td>
<td>R</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Cristofaro Del Prete</td>
<td>R</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol Krizka/kkrizka</td>
<td>L</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Luis Q. T./elezeta</td>
<td>C</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Karol J. Iradier</td>
<td>L</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Alvaro J. Iradier Muro/airadier</td>
<td>L</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Alaoui Youness/kakaroto</td>
<td>A,D,P</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Vivia Nikolaidou</td>
<td>C</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Sander Hoentjen/tjikkun</td>
<td>C</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Philippe Vandenbrouck</td>
<td>P</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>Boris Faure/billiob</td>
<td>B</td>
<td>1041</td>
<td>1041</td>
</tr>
<tr>
<td>Alexander Nestorov</td>
<td>N</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Monaghan</td>
<td>A,D</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Philippe Monaghan - Philip</td>
<td>D</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Philippe Malo</td>
<td>G</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>Danilo Troccoli</td>
<td>G</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>Karel Demeyer/scapor</td>
<td>D</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Tom Jenkins</td>
<td>T</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 7.1: Developers as published on the aMSN website, compared to the extracted developers.
### 7.2. Case study 2: Open source software development

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Developer</td>
</tr>
<tr>
<td>T</td>
<td>Tester</td>
</tr>
<tr>
<td>A</td>
<td>Admin</td>
</tr>
<tr>
<td>I</td>
<td>Internationalization</td>
</tr>
<tr>
<td>P</td>
<td>Project Manager</td>
</tr>
<tr>
<td>R</td>
<td>Retired Developer</td>
</tr>
<tr>
<td>O</td>
<td>Original author of CCMSN (aMSN has been derived from CCMSN)</td>
</tr>
<tr>
<td>PC</td>
<td>PHP Coder</td>
</tr>
<tr>
<td>DB</td>
<td>Database Developer</td>
</tr>
<tr>
<td>PW</td>
<td>PHP Coder for retired website</td>
</tr>
<tr>
<td>PD</td>
<td>Plugin Developer</td>
</tr>
<tr>
<td>Pl</td>
<td>aMSN Plus Developer</td>
</tr>
<tr>
<td>WM</td>
<td>Win Maintainer</td>
</tr>
<tr>
<td>M</td>
<td>Maintainer</td>
</tr>
<tr>
<td>MP</td>
<td>Mac Porter</td>
</tr>
<tr>
<td>GM</td>
<td>Games Plugin Maintainer</td>
</tr>
<tr>
<td>RW</td>
<td>Retired webpage developer</td>
</tr>
<tr>
<td>G</td>
<td>Graphics</td>
</tr>
<tr>
<td>S</td>
<td>Skin Designer</td>
</tr>
<tr>
<td>W</td>
<td>Web Designer</td>
</tr>
<tr>
<td>U</td>
<td>UI Designer</td>
</tr>
<tr>
<td>GA</td>
<td>Graphical Artist</td>
</tr>
<tr>
<td>C</td>
<td>Coder</td>
</tr>
<tr>
<td>Pa</td>
<td>Packager</td>
</tr>
</tbody>
</table>

Table 7.12: Roles of the aMSN developers, as defined on the aMSN website.

### 7.2.2 Conclusion

The development community as extracted from the software repositories of the aMSN project, follows the generals structure of open source software communities, as presented in Figure 7.9. The number of developers with the ‘bug fixer’ role is less than the number of developers with the ‘peripheral developer’ role. A reason for this could be that one of the criteria for being granted access to the Subversion repository, is to actively work on the source code for example.

The criteria for the classification of the developers, as described in Section 7.2.1, required the use of multiple software repositories. Without having multiple software repositories, the classification could not have been made. Thereby, we demonstrated the added value of using multiple software repositories.
4.2 Roles and Structures of the Four OSS Communities

This section analyzes the roles and structures of the four OSS communities we have studied according to the general framework. Except for the Jun project that was initiated in SRA, all the other three projects are a portion of larger OSS communities, therefore, our analysis will be aligned with the whole large communities with special focus on the roles that SRA people play.

Most GNU systems, represented by GCC and Emacs, have a Project Leader who is often the person who initiates the system. In a few GNU systems Project Leaders might be helped by several who do not care about the source code nor be able to understand the code. This breeds the need of supporting those Passive Users.

Members of the Linux Support project at SRA are Peripheral Developers, Bug Fixers, and Readers. They understand the code and are able to find a fix for the bugs, or fix the bugs by themselves when the bugs are reported by customers who are Passive Users.

PostgreSQL does not have a single Project Leader. Instead, it has six Core Members who communicate with each other through a dedicated mailing list to discuss and decide the direction of the system. The inclusion of a new feature must be sponsored by one

\begin{figure}
\centering
\includegraphics[width=\textwidth]{structure.png}
\caption{General structure of an open source software community, as presented in [Nak02]}
\end{figure}

7.3 Conclusion

In the first case study, we demonstrated the applicability of process mining techniques to data from software repositories. We showed that we can get insights in the development process of the TU/e software engineering projects, without having contact the developers involved in the project. In the second case study, we demonstrated the added value of using data from multiple software repositories in combination, as opposed to using data from multiple software repositories in isolation.
Chapter 8

Related Work

In the area of mining software repositories, a considerable amount of work has been done using a single repository. As new types of repositories emerged and developers started to use them, also research on mining data from multiple software repositories has been conducted. We use the term ‘FRASR’ in this chapter for the combination of FRASR with an analysis application, like ProM [Pro09].

8.1 Single software repository

Various types of analysis approaches have been developed using data from a single software repository. This section contains an overview of a number of these approaches, which have been chosen for the availability or existence of a tool.

8.1.1 eROSE

The eROSE plugin for ECLIPSE [Ecl10] guides programmers to related changes [Zim04, Zim05, Ero10]. When a programmer changes a method \( f() \) and method \( f() \) has often been changed together with method \( g() \) in the past, eROSE will suggest the programmer to also change method \( g() \).

eROSE uses information from a CVS repository to find the related methods. By pointing the programmer to methods which should also have to be changed (together with a certain method), the programmer is less likely to miss this other method, resulting in less errors.

The current implementation of FRASR cannot be used to link co-changing methods. FRASR can be used to study the co-evolution of the components or packages, by using the component case in combination with the (advanced) dotted chart of ProM, to get insight in the relation between co-changing packages. This analysis however requires the analyst to find those packages, where eROSE presents the related methods without human interaction.

8.1.2 ProjectWatcher

Gutwin et al. have developed an application named ProjectWatcher [Gut04]. The application can be used to overcome problems related to geographically separated development teams which are working on the same project. One of the issues the application can help to cope with, is finding developers who have the expertise related to an artifact.

ProjectWatcher uses a shadow software repository where fine-grained user modifications are stored. Using this shadow repository, the mining component extracts facts which in
turn are used by the visualization component to provide the information to the developer. The visualization component consists of a plugin for the Eclipse IDE [Ecl10], as shown in Figure 8.1.

![Figure 8.1: An example of the ProjectWatcher plugin in Eclipse [Sch04].](image)

The analysis of a software project using ProjectWatcher may give insights which cannot be gained by FRASR, as ProjectWatcher uses a more fine-grained analysis (up to the level of methods in a class). However, FRASR can also be used to get insights into relations between developers and artifacts. Furthermore, FRASR can be used to get insights into the co-changing files (or packages) in a project, where ProjectWatcher can only provide information about the modifications to a single file over time.

### 8.1.3 Marmoset

Marmoset is a system developed at the University of Maryland. The system supports automated snapshots, submissions and testing of student programming projects [Spa05, Mar10b]. On the one hand the system aims at improving the students programming experience, and on the other hand the system allows for teachers to gain better insights in the way students learn to program.

The system records each intermediate version of a project to a local CVS repository. When doing so, teachers can see more than just the versions handed in by the students (which are to be checked and corrected).

The analysis of the data from the local CVS repositories can be extended by using FRASR (requires adding support for CVS repositories). An analysis application currently supported by Marmoset is FindBugs [Hov04]. FindBugs is applied to each version separately.

### 8.1.4 DynaMine

DynaMine is an application to extract likely error patterns (violations of application-specific coding rules) from data from software repositories [Liv05]. By looking at the incremental changes between revisions, the tool can find highly correlated methods calls (like the eROSE approach) as well as common bug fixes. Using this information, application-specific coding patterns can be discovered. These patterns are then passed to a dynamic analysis tool for validation.
DynaMine can provide insights which cannot be extracted using the current implementation of FRASR. DynaMine for instance can detect method calls that are frequently added to the source code simultaneously. However, FRASR can be used, e.g., to get insights into which developers are related to the artifacts (files for example) having the mined error patterns.

### 8.1.5 SolidSTA

SolidSTA is a commercial application developed by SolidSourceIT [Sol10a]. SolidSTA can connect to Subversion and CVS repositories and visualize trends in activity and contribution patterns. Furthermore, it can visualize trends in software metrics like size and complexity. In order to use the application, the location (URL) and authentication settings of the repository are sufficient. The application supports custom metrics to be calculated via a plugin system. Figure 8.2 shows the architecture of the tool.

![Figure 8.2: The architecture of the SolidSTA tool](image)

A number of analysis aspects for analyzing a software repository, are common for SolidSTA and FRASR. Consider for example the developer activity in terms of number of files added, modified or deleted, and the evolution of files over time (at which points in time were files added and modified etcetera). SolidSTA can however cluster groups of files on metric values (like the McCabe complexity metric for example). FRASR can perform developer matching on aliases from one repository, as developers can use multiple aliases in one repository (for example because credentials were lost).

### 8.1.6 Other approaches

Besides these applications to perform analysis on single software repositories, also several approaches (without an application, which is available) have been proposed.

An other approach which uses sets of frequently changed files, can be used to find traceability links between software artifacts [Kag07]. Having these links is important for various maintenance tasks. This method proposed by Kagdi et al. focusses on relations between artifacts. FRASR can also be used to analyze relations between artifacts, like described in Section 7.1. However, the method described by Kagdi et al. can do this automatically.

The application Xia, developed by Wu et al., uses data from a version control system to perform reverse engineering [Wu04]. Like other applications discussed in this chapter, it contains a plugin for Eclipse [Ecl10] to have an integration with the developers ‘working environment’. FRASR is not integrated in the working environment of the developers. However, FRASR
can be used to get insights into the points in time at which artifacts were modified, where Xia focusses on the differences between different versions of artifacts.

Williams et al. describe a technique which, using data from the version control system of a project, can help to drive and refine the search for bugs [Wil05]. While other applications use static bug finding techniques on a single version, this technique compares the results of all the version to determine when a bug of a specific type was fixed. For instance the absence of checking the return value of a function. FRASR cannot be used for the purposes described by Williams et al.. FRASR can be used to get insights in which developers were involved with the files which contain the bugs, found by the approach by Williams et al..

CVSgrab is an application to gather and visualize data from the CVS version control system [Voi06]. Using this application, questions like “What are the main contributors and their responsibilities?” and “Where were the development issues located, which were discovered and solved during alpha testing some given release?”, can be answered. FRASR can also be used to answer these questions. Furthermore, FRASR can use data from other software repositories, like Subversion or Bugzilla.

8.2 Multiple software repositories

Besides applications which use a single type of software repository as an input, there are also applications available which allow for more types of software repositories to be used. This section contains an overview of these applications.

8.2.1 Hipikat

When joining an existing project, a software developer must get familiar with the project group memory. A project group memory can be seen as a repository of information which group members can use to benefit from past experiences and to cope better with present needs [ˇCub04]. As projects have various software repositories with large amounts of data, it can take a significant amount of time for the developer to go through all this data and find the most relevant parts of it.

To speed up this process, the Hipikat application has been developed [ˇCub05, Hip10]. Hipikat is a plugin for Eclipse [Ecl10]. When connected to the Hipikat server, the plugin can request related artifacts given some artifact (see Figure 8.3 for example), or give the results of a search query for example. The Hipikat server keeps track of all the modifications in the linked repositories.

![Figure 8.3: An example of the use of Hipikat in Eclipse.](image)

Like Hipikat, FRASR can also be used to get insight in data related to some artifact, but this cannot be performed from within Eclipse. However, FRASR can be used to get an overview of the relation between artifacts, over time, with respect to the developers involved, etcetera.
8.2.2 Alitheia Core

As analyzing data from software repositories involves working with large data sets, it can be time consuming to do the pre-processing of all the data, calculating (basic) metrics etcetera. The Alitheia Core application is designed specifically to facilitate software engineering research, without having to perform all the pre-processing steps [Gou09, Ali10]. It allows the researcher to easily extend the application with a plugin to allow for the calculation of custom defined metrics.

Figure 8.4 shows an example of the application. The researcher can easily access the application via the web-based interface.

![Alitheia Core Application](image.png)

**Figure 8.4: An example of the Alitheia Core application.**

Alitheia Core is a more data mining oriented application, compared to the process mining orientation of FRASR. FRASR can for example be used to extract a process model from data from software repositories, where Alitheia Core can only be used to calculate metrics over the data from software repositories. Notice that FRASR has support for some basic metrics like the number of aliases per data source etcetera. Furthermore, export plugins can be implemented for the calculation of other metrics.
8.2.3 softChange

German defined software trails as information left behind by contributors to a development process [Ger04]. This information consists of data from mailing lists, websites, version control systems, etcetera. A history of a project is contained in these software trails, and can be used to recover the evolution of the project.

To analyze these software trails, the softChange application has been developed. softChange imports data from mail archives, CVS and Bugzilla repositories, and has a fact extractor which determines which software trails are related. See Figure 8.5 for the architecture of the application.

![Figure 8.5: The architecture of the softChange application.](image)

The softChange approach mainly focuses on CVS logs. FRASR on the other hand, is usable for a variety of software repositories. Furthermore, as FRASR is not bound to a specific analysis application, it is more flexible.

8.2.4 Other approaches

Wolf et al. propose another approach which uses data from various types of software repositories [Wol09]. In the approach, a social network is constructed around an artifact (a failed build for example). All the developers involved with the artifact are linked to the artifact. By filtering the social network for specific groups of developers (those who have contributed to the code which is a part of the (sub)project for example), a project manager can get insight in the communication behavior of the developers.
8.3 Shortcomings

The applications and approaches described in this chapter have several shortcomings related to the project goal.

As the project goal states that the application should support data from various software repositories, all the single software repository applications are not usable for this purpose. However, it is not always the case that all the available data sources are required to answer an analysis question. These applications can be used in cooperation with FRASR, combining the results from both approaches.

Furthermore, the applications which do support multiple data sources, typically focus on a specific analysis aspect. Where the Hipikat application can be used to find related artifacts, FRASR can also be used to find related developers. Finding related developers is useful when for example the design decisions of an artifact are unclear, and need to be discussed with the developers involved in the decision making process about this artifact.

Even though the softChange approach by German can use data from several data sources, it is mainly about the importing of data from CVS logs. However, German also discusses relevant aspects of mining software repositories in general, like the existence of different developer aliases in and among software repositories.

The Alitheia Core has some features also present in FRASR, like the possibility to define projects and automatically do the preprocessing of data from the software repositories. However, due to the nature of process mining, which requires an event log (with a fixed structure involving cases and tasks etcetera), the metric-centered approach makes it unsuitable to use for process mining.
Chapter 9

Conclusions

Software development is an activity mostly performed in teams. To support software development, software repositories are used for communication and sharing of data. Software repositories are for example bug trackers, version control systems and mailing lists. These software repositories contain valuable data about the execution of the project.

The success of a software project depends on several factors. A number of these factors are related to the development process. Using the historic data from the software repositories, analysis can be performed to get insights into this development process. As the development process consists of temporal related activities, process mining techniques are well suited for this analysis.

The majority of the existing applications for analyzing data from software repositories, is only applicable to one specific type of software repository. As developers use multiple repositories simultaneously, using data from a single repository might provide a limited view on the development processes. See Section 3.2 for more information.

Furthermore, the analysis applications which do support multiple repositories, typically focus on a specific analysis aspect. For example finding related artifacts of an artifact, calculating metrics about the data in the repositories and analyzing trends over time.

In order to be able to perform the types of analysis mentioned before, the goal of this project was formulated as follows:

**Goal of the project:** Create an application which facilitates process analysis of data from various software repositories, in an easy manner.

The application created to comply to the project goal is called FRASR (FRamework for Analyzing Software Repositories). When using FRASR, one can define the way in which the data from the software repositories should be combined to an event log, which then can be used to analyze the data (in an external application).

As the project goal states that the application should support data from various software repositories, FRASR allows to (automatically) match the (different) aliases of the users from the various software repositories. In that way, the data is linked to a single ‘developer’, instead of to all of its aliases.

Furthermore, FRASR supports various methods of combining data from different software repositories, which allows to define the mapping from data to event log elements, without having to write any code.

To demonstrate the added value of FRASR for analyzing software repositories, two case studies have been performed. The first case study (see Section 7.1) shows the applicability
of process mining techniques to data from software repositories, and the possibility to get insight in the development process of a software project, without having to consult the people involved. The second case study (see Section 7.2) shows the added value of using multiple data sources instead of a single data source. This case study analyzes the composition of the development community of an open source software project. As members of some roles of this community can only be identified by looking at data from the version control system and the bug tracker, this analysis could not have been performed by using only a single software repository.

By the case studies, we have demonstrated that FRASR is an application which facilitates process analysis of data from various software repositories, in an easy manner.

9.1 Future work

Although FRASR is a fully functional application which already has been used in several analysis projects, there are still points of improvement. One of the most obvious points of improvement probably, is to support more types of software repositories. Table 9.1 provides an overview of data source types not yet considered by FRASR, which have been used in (large) open source projects, and examples of those projects.

<table>
<thead>
<tr>
<th>Data source type</th>
<th>Used by</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVS [Cvs09]</td>
<td>AWStats [Aws10], Drupal [Dru10]</td>
</tr>
<tr>
<td>Mercurial [Mer10]</td>
<td>Firefox [Bug09], NetBeans [Net10], OpenOffice [Ope10]</td>
</tr>
<tr>
<td>Bazaar [Baz10]</td>
<td>Bugzilla [Bug09], Ubuntu [Ubu10]</td>
</tr>
<tr>
<td>Git [Git10]</td>
<td>Android [And10], jQuery [Jqu10], phpMyAdmin [Php10d]</td>
</tr>
<tr>
<td>SourceForge Thumbs up/down [Sou10]</td>
<td>jEdit [Jed10], WinMerge [Win10b]</td>
</tr>
<tr>
<td>Mantis Bug Tracker [Man10]</td>
<td>phpList [Php10c], SVNKit [Svn10]</td>
</tr>
<tr>
<td>Lauchpad Bug Tracker [Lau10]</td>
<td>DC++ [Dcp10], VLC [Vid10]</td>
</tr>
<tr>
<td>phpBB Forum [Php10b]</td>
<td>ArgoUML [Arg10], Audacity [Aud10], FileZilla [Fil10], WinMerge [Win10b]</td>
</tr>
<tr>
<td>Google Groups [Goo10]</td>
<td>Firefox [Fir10], Joomla [Joo10]</td>
</tr>
</tbody>
</table>

Table 9.1: Overview of possible data source type extensions to FRASR.

Besides adding support for other data source types, there are also points of improvement in the current components of the application. The current implementation of the cache stores the downloaded (and unparsed) files on the disk. When using the data from the cache, the data is parsed every time it is used. A more efficient implementation in which the parsed data is stored (in a relational database for example) can solve this problem.

Exporting a case mapping in FRASR can only be done one at a time. When having multiple case mappings defined, a feature to define batch exports can reduce the number of times the user has to interact with FRASR (and wait in between).

Several data source types in FRASR download all the items (bug reports, or mail items) one after each other. In order to speed up this process, multiple items can be downloaded simultaneously, decreasing the overall time to download all the data. However, some servers
may not support clients with several open connections. Notice that when using the cache functionality, this issue only applies to the initial fetching of data.

Applications like ProM [Pro09] have support for log events with ‘special’ attributes, in addition to attributes like the timestamp, originator and event name. In ProM the Event Data Attribute Visualizer plugin can visualize data in the special attributes.

The analysis questions of the case studies could not always be answered with maximal certainty. In order to quickly navigate to the ‘data items’ in the data sources where the events originated from, a special attribute can provide a direct link to this data item. This link can point to a single bug report or an email message.

9.2 Future analysis projects

Besides the analysis projects described in Chapter 7, several other analysis projects can benefit from the potential of FRASR (in combination with an external analysis application like ProM). This section provides examples of such analysis projects.

9.2.1 Conformance checking hour registrations

Project managers use data from hour registrations (which developer has worked on which part of the system etcetera) to estimate whether the (next) deadline will be made [Let05]. Therefore, it is important that such hour registrations are an accurate representation of the development process.

Manually checking all these hour registrations with the involved developers is a time consuming activity. To reduce the time to check the hour registrations, FRASR can be used. Using FRASR, the data from the hour registrations can be combined with the data from the software repositories of the project. Notice that this may require implementing a new data source. The combined data can then be analyzed. The project manager now only has to consult the developers for which the data showed significant deviations.

9.2.2 Separation of concerns

In an organization, there can be various departments. In software development, there can be various teams [Saw04]. For example a development team, a test team and a maintenance team. Each of these teams has a specific role in the development process. For the quality of the development process, it is important that there is a separation of concerns between the various development teams. The test team should for example only test code which is created by an other team.

To analyze these properties, FRASR can be used in combination with the organizational miner plugin in ProM [Sou08]. Notice that techniques exist to classify whether a revision from a version control is a bug fix, a feature addition, etcetera [Has08].

When having the results of the analysis, one can check whether developers from the test team for example only report bugs, but do not fix them. Furthermore, developers from the maintenance team probably should only fix bugs, but do not add new functionality.

9.2.3 Social network analysis

Bird et al. describe an approach to mine social networks from email conversations [Bir06]. The extracted social networks are compared with social networks from a version control system
(CVS). Questions which can be answered using this approach are:

- What are the properties of the social network of developers?
- Are developers who send a lot of messages on the mailing list also very active in source code change?
- Do developers play a different role than non-developers?
- Do the most active developers have the highest status among developers?

The analysis Bird et al. works with two data sources: a mailing list archive and a version control system. Nowadays, developers use not only mailing lists and version control systems, but also discussion boards, bug reports and wiki articles for example. FRASR can be used to analyze all these data sources.

Analyzing the social network of a development team is important, as the development of software is social activity [Cai96]. Using social network analysis, the relationship between team structure and team performance can be analyzed [Yan04].

9.2.4 Component focus

Before releasing (a version of) a software product, it is important that all the components are of high quality [God09]. When there are components which with a lower quality, the product can only be released when the quality of these components is raised sufficiently. An indication of the quality of a component can be the number of bugs posted concerning the component, the number of messages posted about the component etc.

The identification of ‘problematic components’ can be done using FRASR. Using this information, more resources could be assigned to this component in order to make the deadline.

9.2.5 Single point of failure

In a development team, besides common tasks, there can be tasks which are typically performed by a single developer. An advantage of such approach is that the developer has the most knowledge about that task. However, when this developer is not available (due to illness for example), the knowledge about the task is also not available. In this setting, a task can be a component or a module of the source code of a system for example. This can lead to a problem when this component or module has to be changed for example. Therefore, it is important to know (before the developer becomes unavailable), which of the components or modules have such a dependency.

In order to detect such a dependency, FRASR can be used in combination with the ‘originator by task matrix’ plugin of ProM [Med08].
# List of Figures

2.1 The addition of information to an artifact in a centralized version control system. 4
2.2 The lifecycle of a bug report in Bugzilla. 5
2.3 Process mining. 7
2.4 Process mining meta model. 7
2.5 A three dimensional view of a workflow. 8
2.6 A mined workflow. 9
2.7 The interface of ProM 6 showing the summary of an event log. 9
2.8 An example of the advanced dotted chart plugin in ProM. 11
3.1 The relation of FRASR with data sources and analysis applications. 13
4.1 FRASR object model. 18
4.2 FRASR data sources object model. 20
4.3 FRASR case mappings object model. 21
4.4 Internal log object model used in FRASR. 25
4.5 An example of the merging process at the developer matching. 26
6.1 Decreasing recall for increasing threshold value. 36
6.2 Decreasing match percentage for increasing threshold value. 37
7.1 The method used to perform the case studies. 39
7.2 V-model for software development. 41
7.3 Overlap between project phases of the Prifes project. 43
7.4 The events of the project manager regarding the agendas and the SPMP, per project phase. 45
7.5 The Prifes source files used for the prototype and the final implementation. 47
7.6 The Diffusion source code files. 48
7.7 Developer activity of the aMSN project. 53
7.8 Developer activity of the aMSN project, sorted by the number of events. 54
7.9 General structure of an open source software community. 60
8.1 An example of the ProjectWatcher plugin in Eclipse. 62
8.2 The architecture of the SolidSTA tool. 63
8.3 An example of the use of Hipikat in Eclipse. 64
8.4 An example of the Altheia Core application. 65
8.5 The architecture of the softChange application. 66
B.1 A view of FRASR where no project is loaded. 89
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.2</td>
<td>Manage data sources view of FRASR.</td>
<td>90</td>
</tr>
<tr>
<td>B.3</td>
<td>Manage cache view of FRASR.</td>
<td>91</td>
</tr>
<tr>
<td>B.4</td>
<td>Project information view of FRASR.</td>
<td>92</td>
</tr>
<tr>
<td>B.5</td>
<td>Project case mappings view of FRASR.</td>
<td>93</td>
</tr>
<tr>
<td>B.6</td>
<td>Project data source filters view of FRASR.</td>
<td>93</td>
</tr>
<tr>
<td>B.7</td>
<td>Project developer matching view of FRASR.</td>
<td>94</td>
</tr>
<tr>
<td>B.8</td>
<td>Project export view of FRASR.</td>
<td>95</td>
</tr>
<tr>
<td>B.9</td>
<td>Project settings view of FRASR.</td>
<td>96</td>
</tr>
</tbody>
</table>
## List of Tables

2.1 An example of an event log. ........................................... 8
4.1 The data sources currently supported by FRASR. ............... 19
4.2 Variants of the datetime case in FRASR. ........................ 22
4.3 Default developer matching settings of FRASR. ................. 29
4.4 Details of the first developer matching example. ............. 29
4.5 Details of the second developer matching example. .......... 30
5.1 Decrease in time exporting data from SourceForge projects, using the cache. 32
7.1 Results for other software engineering projects, showing similar results. . . 44
7.2 Results for other software engineering projects, related to the performance of the project manager. ................................. 46
7.3 Data related to developer: -null-/null-. ............................ 54
7.4 Developers assigned to each role. ................................ 55
7.5 Developers classified as bug reporter. ............................ 55
7.6 Developers classified as bug fixer. ................................ 55
7.7 Developers classified as peripheral developer. .................... 56
7.8 Developers classified as active developer. ....................... 56
7.9 Developers classified as core member. ............................ 56
7.10 Developers classified as project leader. .......................... 57
7.11 Developers as published on the aMSN website, compared to the extracted developers. ................................................. 58
7.12 Roles of the aMSN developers, as defined on the aMSN website .... 59
9.1 Overview of possible data source type extensions to FRASR. .... 70
A.1 Execution times for FRASR performing 101 developer matching calculations per project. .............................................. 83
A.2 Data sources of aMSN used for the developer matching experiments. .... 84
A.3 Data sources of Gallery used for the developer matching experiments. .... 84
A.4 Data sources of jEdit used for the developer matching experiments. .... 85
A.5 Data sources of Notepad++ used for the developer matching experiments. .... 85
A.6 Data sources of PhpMyAdmin used for the developer matching experiments. .... 86
A.7 Data sources of ScummVM used for the developer matching experiments. .... 86
A.8 Data sources of WinMerge used for the developer matching experiments. .... 87

75
Bibliography


Appendix A

Developer Matching Experiments

This appendix contains the details of the developer matching experiments, described in Chapter 6. Each section contains the details of one of the projects considered for the experiments. These details include the settings used in FRASR, as well as the number of developer aliases extracted per data source.

Developer aliases are extracted per data source. For each data element (a comment on a bug report, a reply to a mailing list message, a revision in a version control system, etcetera), the associated IdInfo, NameInfo and MailInfo objects are extracted. When for that data source there does not already exists a developer alias with those objects, a new developer alias is created.

Table A.1 presents the execution times\(^1\) for the calculation of the results for the experiments. The load times are the times to load the developer aliases from the data sources (see Table 5.1 in Section 5.1 for more information about the number of revision and bugs etcetera). The execution times are the times to execute the developer matchings for the 101 experiments.

<table>
<thead>
<tr>
<th>Project</th>
<th>Date registered</th>
<th># aliases</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>aMSN</td>
<td>2002-05-22</td>
<td>2715</td>
<td>74 min</td>
</tr>
<tr>
<td>Gallery</td>
<td>2000-06-18</td>
<td>4682</td>
<td>236 min</td>
</tr>
<tr>
<td>jEdit</td>
<td>1999-12-06</td>
<td>5604</td>
<td>394 min</td>
</tr>
<tr>
<td>Notepad++</td>
<td>2003-11-24</td>
<td>9838</td>
<td>1038 min</td>
</tr>
<tr>
<td>PhpMyAdmin</td>
<td>2001-03-18</td>
<td>10615</td>
<td>1234 min</td>
</tr>
<tr>
<td>ScummVM</td>
<td>2001-10-05</td>
<td>4316</td>
<td>179 min</td>
</tr>
<tr>
<td>WinMerge</td>
<td>2000-10-20</td>
<td>1669</td>
<td>31 min</td>
</tr>
</tbody>
</table>

Table A.1: Execution times for FRASR performing 101 developer matching calculations per project.

Notice when performing this analysis at a later point in time, other results can be obtained, as new developers will join the projects and new data sources will be created.

\(^1\) All the experiments have been performed on a 32bit Windows7 machine with an Intel Core2 Quad CPU @ 2.40 GHz with 3GB of memory, using the TU/e internet connection.
## Appendix A. Developer Matching Experiments

### A.1 aMSN

Table A.2 presents the settings used in FRASR for the developer matching experiment concerning aMSN [Ams10].

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>FRASR settings</th>
<th># aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF bug repository</td>
<td>Bugs</td>
<td>Group Id: 54091, Tracker Id: 472655</td>
<td>1537</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Feature requests</td>
<td>Group Id: 54091, Tracker Id: 472658</td>
<td>229</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Patches</td>
<td>Group Id: 54091, Tracker Id: 472657</td>
<td>103</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Plugins</td>
<td>Group Id: 54091, Tracker Id: 733148</td>
<td>79</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Skins</td>
<td>Group Id: 54091, Tracker Id: 813041</td>
<td>52</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Support requests</td>
<td>Group Id: 54091, Tracker Id: 472656</td>
<td>147</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Translations</td>
<td>Group Id: 54091, Tracker Id: 813042</td>
<td>8</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Commits</td>
<td>Mailing List Id: amsn-commits</td>
<td>35</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Devel</td>
<td>Mailing List Id: amsn-devel</td>
<td>271</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Lang</td>
<td>Mailing List Id: amsn-lang</td>
<td>205</td>
</tr>
<tr>
<td>SVN</td>
<td></td>
<td>Url: <a href="https://amsn.svn.sourceforge.net/svnroot/amsn/">https://amsn.svn.sourceforge.net/svnroot/amsn/</a></td>
<td>49</td>
</tr>
</tbody>
</table>

Total: 2715

Table A.2: Data sources of aMSN used for the developer matching experiments.

### A.2 Gallery

Table A.3 presents the settings used in FRASR for the developer matching experiment concerning Gallery [Gal10].

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>FRASR settings</th>
<th># aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF bug repository</td>
<td>Bugs</td>
<td>Group Id: 7130, Tracker Id: 107130</td>
<td>1949</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Feature requests</td>
<td>Group Id: 7130, Tracker Id: 357130</td>
<td>1335</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Patches</td>
<td>Group Id: 7130, Tracker Id: 307130</td>
<td>355</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Translations</td>
<td>Group Id: 7130, Tracker Id: 582564</td>
<td>308</td>
</tr>
<tr>
<td>TRAC Tickets</td>
<td></td>
<td>Url: <a href="http://sourceforge.net/apps/trac/gallery/report/6">http://sourceforge.net/apps/trac/gallery/report/6</a></td>
<td>235</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Announce</td>
<td>Mailing List Id: gallery-announce</td>
<td>10</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Checkins</td>
<td>Mailing List Id: gallery-checkins</td>
<td>42</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Devel</td>
<td>Mailing List Id: gallery-devel</td>
<td>365</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Translations</td>
<td>Mailing List Id: gallery-translations</td>
<td>25</td>
</tr>
<tr>
<td>SVN</td>
<td></td>
<td><a href="https://gallery.svn.sourceforge.net/svnroot/gallery/">https://gallery.svn.sourceforge.net/svnroot/gallery/</a></td>
<td>60</td>
</tr>
</tbody>
</table>

Total: 4682

Table A.3: Data sources of Gallery used for the developer matching experiments.
### A.3 jEdit

Table A.4 presents the settings used in FRASR for the developer matching experiment concerning jEdit [Jed10].

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>FRASR settings</th>
<th># aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF bug repository</td>
<td>Bugs</td>
<td>Group Id: 588, Tracker Id: 100588</td>
<td>1612</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Feature requests</td>
<td>Group Id: 588, Tracker Id: 350588</td>
<td>265</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>jEditLauncher bugs</td>
<td>Group Id: 588, Tracker Id: 566996</td>
<td>55</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Merge requests</td>
<td>Group Id: 588, Tracker Id: 1235750</td>
<td>15</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Patches</td>
<td>Group Id: 588, Tracker Id: 300588</td>
<td>302</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Plugin bugs</td>
<td>Group Id: 588, Tracker Id: 565475</td>
<td>791</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Plugin central submission</td>
<td>Group Id: 588, Tracker Id: 625093</td>
<td>169</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Plugin feature requests</td>
<td>Group Id: 588, Tracker Id: 997936</td>
<td>143</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Print patches</td>
<td>Group Id: 588, Tracker Id: 997937</td>
<td>90</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Announce</td>
<td>Mailing List Id: jedit-announce</td>
<td>8</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Cvs</td>
<td>Mailing List Id: jedit-cvs</td>
<td>144</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Devel</td>
<td>Mailing List Id: jedit-devel</td>
<td>586</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Users</td>
<td>Mailing List Id: jedit-users</td>
<td>1311</td>
</tr>
<tr>
<td>SVN</td>
<td></td>
<td>Url: <a href="https://jedit.svn.sourceforge.net/svnroot/jedit/">https://jedit.svn.sourceforge.net/svnroot/jedit/</a></td>
<td>113</td>
</tr>
</tbody>
</table>

**Total:** 5604

Table A.4: Data sources of jEdit used for the developer matching experiments.

### A.4 Notepad++

Table A.5 presents the settings used in FRASR for the developer matching experiment concerning Notepad++ [Not10].

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>FRASR settings</th>
<th># aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF bug repository</td>
<td>Bugs</td>
<td>Group Id: 95717, Tracker Id: 612382</td>
<td>1686</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Feature requests</td>
<td>Group Id: 95717, Tracker Id: 612385</td>
<td>824</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Patches</td>
<td>Group Id: 95717, Tracker Id: 612384</td>
<td>227</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Support requests</td>
<td>Group Id: 95717, Tracker Id: 612383</td>
<td>72</td>
</tr>
<tr>
<td>SF forum</td>
<td>Boycott Notepad++ here</td>
<td>Project Id: notepad-plus, Forum Id: 813349</td>
<td>119</td>
</tr>
<tr>
<td>SF forum</td>
<td>Deutsch forum</td>
<td>Project Id: notepad-plus, Forum Id: 731535</td>
<td>398</td>
</tr>
<tr>
<td>SF forum</td>
<td>Foro en espanol</td>
<td>Project Id: notepad-plus, Forum Id: 956984</td>
<td>35</td>
</tr>
<tr>
<td>SF forum</td>
<td>Forum em portugues</td>
<td>Project Id: notepad-plus, Forum Id: 809450</td>
<td>71</td>
</tr>
<tr>
<td>SF forum</td>
<td>Forum francais</td>
<td>Project Id: notepad-plus, Forum Id: 731514</td>
<td>370</td>
</tr>
<tr>
<td>SF forum</td>
<td>Help</td>
<td>Project Id: notepad-plus, Forum Id: 33175</td>
<td>2622</td>
</tr>
<tr>
<td>SF forum</td>
<td>Open discussion</td>
<td>Project Id: notepad-plus, Forum Id: 331753</td>
<td>2451</td>
</tr>
<tr>
<td>SF forum</td>
<td>Plugin development</td>
<td>Project Id: notepad-plus, Forum Id: 482781</td>
<td>715</td>
</tr>
<tr>
<td>SF forum</td>
<td>Translation</td>
<td>Project Id: notepad-plus, Forum Id: 558104</td>
<td>64</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Notepad++</td>
<td>Mailing List Id: notepad-plus-plus</td>
<td>180</td>
</tr>
<tr>
<td>SVN</td>
<td></td>
<td>Url: <a href="https://notepad-plus.svn.sourceforge.net/svnroot/notepad-plus">https://notepad-plus.svn.sourceforge.net/svnroot/notepad-plus</a></td>
<td>4</td>
</tr>
</tbody>
</table>

**Total:** 9838

Table A.5: Data sources of Notepad++ used for the developer matching experiments.
### A.5 PhpMyAdmin

Table A.6 presents the settings used in FRASR for the developer matching experiment concerning PhpMyAdmin [Php10d].

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>FRASR settings</th>
<th># aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF bug repository</td>
<td>Bugs</td>
<td>Group Id: 23067, Tracker Id: 377408</td>
<td>2381</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Feature requests</td>
<td>Group Id: 23067, Tracker Id: 377411</td>
<td>913</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Patches</td>
<td>Group Id: 23067, Tracker Id: 377410</td>
<td>518</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Support requests</td>
<td>Group Id: 23067, Tracker Id: 377409</td>
<td>1221</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Themes</td>
<td>Group Id: 23067, Tracker Id: 689412</td>
<td>105</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Translations</td>
<td>Group Id: 23067, Tracker Id: 387654</td>
<td>282</td>
</tr>
<tr>
<td>SF forum</td>
<td>Help</td>
<td>Project Id: phpmyadmin, Forum Id: 72909</td>
<td>4064</td>
</tr>
<tr>
<td>SF forum</td>
<td>Hilfe</td>
<td>Project Id: phpmyadmin, Forum Id: 297172</td>
<td>401</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Devel</td>
<td>Mailing List Id: phpmyadmin-devel</td>
<td>325</td>
</tr>
<tr>
<td>SF email archive</td>
<td>News</td>
<td>Mailing List Id: phpmyadmin-news</td>
<td>9</td>
</tr>
<tr>
<td>SF email archive</td>
<td>SVN</td>
<td>Mailing List Id: phpmyadmin-svn</td>
<td>3</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Translators</td>
<td>Mailing List Id: phpmyadmin-translators</td>
<td>42</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Trk-bugs</td>
<td>Mailing List Id: phpmyadmin-trk-bugs</td>
<td>1</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Trk-featreq</td>
<td>Mailing List Id: phpmyadmin-trk-featreq</td>
<td>1</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Trk-patches</td>
<td>Mailing List Id: phpmyadmin-trk-patches</td>
<td>1</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Trk-support</td>
<td>Mailing List Id: phpmyadmin-trk-support</td>
<td>1</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Trk-translat</td>
<td>Mailing List Id: phpmyadmin-trk-translat</td>
<td>1</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Users</td>
<td>Mailing List Id: phpmyadmin-users</td>
<td>318</td>
</tr>
<tr>
<td>SVN</td>
<td></td>
<td>Url: <a href="https://phpmyadmin.svn.sourceforge.net/svnroot/phpmyadmin/">https://phpmyadmin.svn.sourceforge.net/svnroot/phpmyadmin/</a></td>
<td>28</td>
</tr>
</tbody>
</table>

**Total:** 10615

Table A.6: Data sources of PhpMyAdmin used for the developer matching experiments.

### A.6 ScummVM

Table A.7 presents the settings used in FRASR for the developer matching experiment concerning ScummVM [Scu10].

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>FRASR settings</th>
<th># aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF bug repository</td>
<td>Bugs</td>
<td>Group Id: 37116, Tracker Id: 418820</td>
<td>2833</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Feature requests</td>
<td>Group Id: 37116, Tracker Id: 418823</td>
<td>508</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Patches</td>
<td>Group Id: 37116, Tracker Id: 418822</td>
<td>539</td>
</tr>
<tr>
<td>SF email archive</td>
<td>CVS logs</td>
<td>Mailing List Id: scummvm-cvs-logs</td>
<td>50</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Devel</td>
<td>Mailing List Id: scummvm-devel</td>
<td>246</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Tracker</td>
<td>Mailing List Id: scummvm-tracker</td>
<td>4</td>
</tr>
<tr>
<td>SVN</td>
<td></td>
<td>Url: <a href="https://scummvm.svn.sourceforge.net/svnroot/scummvm/">https://scummvm.svn.sourceforge.net/svnroot/scummvm/</a></td>
<td>136</td>
</tr>
</tbody>
</table>

**Total:** 4316

Table A.7: Data sources of ScummVM used for the developer matching experiments.
A.7 WinMerge

Table A.8 presents the settings used in FRASR for the developer matching experiment concerning WinMerge [Win10b].

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>FRASR settings</th>
<th># aliases</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF bug repository</td>
<td>Bugs</td>
<td>Group Id: 13216, Tracker Id: 113216</td>
<td>741</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Feature requests</td>
<td>Group Id: 13216, Tracker Id: 363216</td>
<td>692</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Support requests</td>
<td>Group Id: 13216, Tracker Id: 213216</td>
<td>79</td>
</tr>
<tr>
<td>SF bug repository</td>
<td>Todo</td>
<td>Group Id: 13216, Tracker Id: 759426</td>
<td>26</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Announce</td>
<td>Mailing List Id: winmerge-announce</td>
<td>3</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Development</td>
<td>Mailing List Id: winmerge-development</td>
<td>4</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Support</td>
<td>Mailing List Id: winmerge-support</td>
<td>86</td>
</tr>
<tr>
<td>SF email archive</td>
<td>SVN</td>
<td>Mailing List Id: winmerge-svn</td>
<td>1</td>
</tr>
<tr>
<td>SF email archive</td>
<td>Translate</td>
<td>Mailing List Id: winmerge-translate</td>
<td>3</td>
</tr>
<tr>
<td>SF email archive</td>
<td>User</td>
<td>Mailing List Id: winmerge-user</td>
<td>15</td>
</tr>
<tr>
<td>SVN</td>
<td></td>
<td>Url: <a href="https://winmerge.svn.sourceforge.net/svnroot/winmerge">https://winmerge.svn.sourceforge.net/svnroot/winmerge</a></td>
<td>19</td>
</tr>
</tbody>
</table>

Total: 1669

Table A.8: Data sources of WinMerge used for the developer matching experiments.
Appendix B

FRASR GUI

As FRASR should be easy to use, a GUI has been build in which all the necessary actions can be performed to export data from software repositories to a format which is supported by the chosen analysis application. Figure B.1 shows the view of FRASR when no project is loaded. In the menu bar, options to export or import settings are provided. The tabs below the menu bar (Project, Manage data sources, Manage cache and Application log) can be used to navigate to their respective pages. The part on the left (Data sources) contains the list of data sources defined in the application. The part on the right shows the application log. The application log shows the actions performed in the application. The status bar at the bottom of the application shows information about the currently loaded project, and indicates whether the application is busy loading data for example (not visible in the figure).

The discussion of the different views in FRASR are ordered by their place in the workflow of using FRASR, see Figure 7.1 in Chapter 7. Notice that in the GUI, the tabs for the views are ordered differently. This ordering has been chosen since the Project tab is used more often compared to the other tabs for example.

B.1 Manage data sources

Figure B.2 shows FRASR with a project loaded (PhpMyAdmin [Php10d]). The part on the left shows the data sources categorized per category (version control system, bug tracker,
mailing list, etcetera) and per data source type (Subversion, Bugzilla, etcetera). The part on the right shows panels to add data sources, modify data sources and delete data sources.

Figure B.2: Manage data sources view of FRASR.

B.2 Manage cache

Figure B.3 shows the manage cache view of FRASR, where the PhpMyAdmin project is loaded. The select box at the top lists all the data sources in the application which have support for a cache. The current implementation of the Mbox mails data source for example only works on locally stored files, so the (file) cache would be unnecessary in this case, as there is no decrease in loading time for the data from this data source. The tree view in the center shows the current content of the cache, in this case an HTML file for each bug report (the numbers indicate the bug id). The action buttons can be used to perform management actions on the cache. The modify settings form can be used to update which data is used per data source (cache, source, or data from the cache supplemented by data from the source) and the whether new data is added to the cache.
B.3. Project

When the data sources have been defined (or imported) to FRASR, they can be used in a project. The project view of FRASR consists of a number of tabs, which are further explained in the following sections.

B.3.1 Information tab

Figure B.4 shows the PhpMyAdmin project. The information panel shows the data sources linked to the project, the statistics about the project and a panel to modify the project name and comments.
Appendix B. FRASR GUI

Figure B.4: Project information view of FRASR.

B.3.2 Case mappings tab

The view in which the case mappings can be defined and modified is presented in Figure B.5. Three case mappings are defined (Artifacts, Subversion and Weeks). The name and type of a case mapping can be modified on the case tab on the right. Using the binding tab, the data sources can be linked to a case mapping.

B.3.3 Filters tab

Figure B.6 shows the data source filters view of FRASR. The tree on the left shows the defined projects per data source. The panel on the right can be used to add a new filter. Notice that the first filter (revision > 500) keeps all the revisions which have a revision id larger than 500 and ignores all the other revisions.
B.3. Project

Figure B.5: Project case mappings view of FRASR.

Figure B.6: Project data source filters view of FRASR.
B.3.4 Developers tab

The tab in which all the actions concerning the developer matching can be performed is shown in Figure B.7. The tree on the left shows the developers in the project (the PhpMyAdmin project). For each developer, the linked developer aliases are visible as children. For each linked developer alias, the data source from which it originated is visible after the name. With the L button in the middle, this developer alias can be selected (located) in the tree on the right. The other two buttons in the middle can be used to link developer aliases to developers. The tree on the right shows the developer aliases of the selected data source (from the project). The circle in front of the name indicates whether this developer alias is linked to a developer (green indicates that there is a link, red indicates that there is no link).

The buttons below the developer tree can be used to delete a developer, or all developers, and to expand or collapse the tree. Below those buttons, two panels are visible in which developers can be added (and modified) manually. The actions panel can be used to reload (or clear) the developer aliases from the selected data source (or all data sources) and to calculate the developer matching. Notice when calculating the developer matching, the current matching is deleted first. At the bottom of the figure, statistics about the developer matching are presented.

Figure B.7: Project developer matching view of FRASR.
B.3.5 Export tab

Figure B.8 shows the project export view of FRASR. In this view, the export format can be selected, as well as the case mapping to export. Furthermore, it is possible to anonymize the developer matching (resulting in names like Developer 1, Developer 2, etcetera for the developers). Notice that these names are only changed in the export, in the project, the original names are kept. Furthermore, it is possible to indicate that while exporting the data, the developer matching must be recalculated. In case that there is no developer matching present in the project, this option has to be selected in order to get different developer names (otherwise, all the events get the same, default, developer name).

Furthermore, the export filters (restricting the event to a certain period in time) can be defined and modified at this view.

![Image of project export view of FRASR](image)

Figure B.8: Project export view of FRASR.
B.3.6 Settings tab

The settings with respect to the data extraction and developer matching can be modified at the project settings tab (Figure B.9). All the settings for the developer matching (except the minimum combine length) are reals ranging from 0 to 1 (included).

![Project settings view of FRASR.](image)

Figure B.9: Project settings view of FRASR.
Appendix C

Case Study Details

This appendix contains detailed information about the case studies, as described in Chapter 7. All the case studies have been performed on a 32bit Windows7 machine with an Intel Core2 Quad CPU @ 2.40 GHz with 3GB of memory, using the TU/e internet connection.

C.1 Case study 1: TU/e Software Engineering Project

The data sources which have been used for the first case study belong to the software engineering projects of the TU/e [Sep10]. Notice that the data sources from all the projects are not publicly available due to confidentiality reasons.

The time for exporting the data from the software repositories and creating the event logs, took between 10 seconds and 4 minutes for the various software engineering projects. The developer matching took about 5 minutes for each project, as the SPMP has to be consulted to get the names of the students etcetera.

**Diffusion:** The only available data source for the Diffusion project is a Subversion repository with 1134 revisions. The developer matching for this project was manually constructed, resulting in a separate developer for each developer alias from the Subversion repository. The number of developer aliases from the Subversion repository (8) matched the number of developers mentioned in the Software Project Management Plan of the project. The data from the Diffusion project ranges from February 12, 2007 until July 25, 2007.

**Dempsey:** The only available data source for the Dempsey project is a Subversion repository with 2082 revisions. The developer matching constructed for this project has a separate developer for each alias in the repository, matching the people involved in the project, as described in the SPMP. The data from the Dempsey project ranges from December 6, 2004 until June 9, 2005.

**DAVIS:** The only available data source for the DAVIS project is a Subversion repository with 1069 revisions. The developer matching constructed for this project has a separate developer for each alias in the repository, matching the people involved in the project, as described in the SPMP. The data from the DAVIS project ranges from December 9, 2005 until September 14, 2006. An inspection revealed that the project file in the development library was updated after a period of three months in which there were no events.
Appendix C. Case Study Details

Prifes: The available data sources of the Prifes project are:

- Subversion: 2032 revisions.
- Trac tickets: 234 tickets, 505 modifications/comments/...
- Trac Wiki: 71 articles, 178 modifications.
- Mails: 73 threads, 124 messages.

As I was personally involved in this project, the developer matching has been constructed manually resulting in a perfect matching. The data from the Prifes project ranges from January 30, 2008 until October 30, 2008. This period of nine months suggests that the project exceeded the six month period available for this project. However, the mailing list was used for other purposes after the project.

PRINCESS: The available data sources of the Prifes project are:

- Subversion: 1574 revisions.
- Trac tickets: 20 tickets, 51 modifications/comments/...
- Trac Wiki: 66 articles, 204 modifications.
- Mails: 130 threads, 204 messages.

As I was personally involved in this project, the developer matching has been constructed manually resulting in a perfect matching. The data from the PRINCESS project ranges from January 19, 2009 until July 9, 2009.

Octosep: The available data sources of the Octosep project are:

- Subversion: 1553 revisions.
- Mails: 218 threads, 409 messages.

The developer matching was constructed using information from a project member, resulting in a perfect matching. The data from the Octosep project ranges from January 27, 2009 until July 20, 2009.

C.1.1 Question 1: How strictly has the V-development model been applied?

The case mapping used for this question is the component case. For each available data source, the associated data source binding has been used.

The component case settings for each event binding are chosen in such a way that the the case instances of all the extracted events are from the following set of event names: URD, SRD, ADD, DDD, STD, SPMP, UTP, ITP, STP, ATP, SVP, SQAP, SCMP, Code, Presentation, Minutes, Agenda, ProcessReport. These names are the abbreviations of the project documents defined in the ESA software engineering standards [ESA91, Jon97]. All event which cannot be mapped onto one of the events from this set, are assigned the -null- value.

The settings described for each data source apply to the Prifes project. The other software engineering projects had similar settings, customized to fit the specific data sources. For example the path names in the Subversion repositories.
Mails + Trac wiki

The bindings for the mails and Trac wiki data sources, use keywords to assign the data elements to case instances. As some of the data elements for the mails and Trac wiki data sources contain synonyms, these are added to the keywords parameter in FRASR. The chosen keywords are: URD, SRD, ADD, DDD, STD, SPMP, UTP, ITP, STP, ATP, SVVPAT, SVVPUT, SVVPIT, SVVPST, SVVP, SQAP, SCMP, code, presentatie, notulen, agenda, process report, processreport. Notice that in FRASR, the white spaces between the keywords have been left out. In case these white spaces would have been a part of the list of keywords, the white spaces would also have to be matched in the body of a mailing list message for example.

In order to have a fixed set of case identifiers (given at Section 7.1.1), some keywords have to be replaced. The following list contains those replacements.

- process report:ProcessReport
- processreport:ProcessReport
- notulen:Minutes
- agenda:Agenda
- code:Code
- presentatie:Presentation
- SVVPIT:ITP
- SVVPUT:UTP
- SVVPAT:ATP
- SVVPST:STP

Subversion

The following list contains the regular expressions used to extract the case identifier of a path name. Recall that only the first regular expression which has a match is applied.

- .*srd.*:SRD
- .*add.*:ADD
- .*urd.*:URD
- .*ddd.*:DDD
- .*svvpat.*:ATP
- .*svvput.*:UTP
- .*svvpst.*:STP
- .*svvput.*:ITP
- .*spmp.*:SPMP
- .*svvp.*:SVVP
- .*scmp.*:SCMP
- .*sqap.*:SQAP
- .*std.*:STD
- .*sum.*:SUM
- */\java:Code
- */\jar:Code
- */\src.*:Code
Appendix C. Case Study Details

- `*.ppt:Presentation`
- `*.*/presentatie.+:Presentation`
- `*.*/interim.+:Presentation`
- `*.*/processreport.+:ProcessReport`
- `*.*/Progress reports.+:SPMP`
- `*.*/Notulen/.+:Minutes`
- `*.*/Agenda/.+:Agenda`

**Trac tickets**

The data field used as the case identifier is the `component` field. The regular expressions to normalize the set of case identifiers are:

- `server:Code`
- `clientserver:Code`
- `printer:Code`
- `overige:-null-`

The first three items correspond to components from the architecture of the system designed in the Project. The fourth item considers tickets not belonging to one of those components.

**C.1.2 Question 2: Did the project manager perform his tasks?**

For this question, the same settings have been used as for the first question (see Appendix C.1.1). Additionally, the event log has been filtered to only keep the events from the project manager. The names of the project managers were extracted from SPMP documents in the Subversion repositories.

**C.1.3 Question 3: Was the prototype a part of the final implementation?**

The case mapping used for this question is the data source field case. The Subversion repository has been linked as the only data source, having the modification `- itemName` as the case instance. The events are created by a detailed binding, using only the events from the level of the modifications, with the modification type as the event name.

**Prifes**

The filters used to filter out all the non-source code related paths are:

- `/(doc|archive|internal|resources|release|other).*`
- `^[^-\.]`
- `.*[^^-\.]`
- `.*\.nl_8084`

All the case instances of which the case identifier matches one these regular expressions, are filtered out. Notice that the fourth item filters out generated code.
Diffusion

The filters used to filter out all the non-source code related paths are:

- /\(archive\)\|\(master\)\|\(development\)/deliverables\)\|\(development\)/presentations\).*
- /\((development\)/documents\)\|\(development\)/code\)/doxygen\)test\).*
- .*/[^\.

All the case instances of which the case identifier matches one these regular expressions, are filtered out.

Other projects

The filters used for other projects are similar to those of Prifes and Diffusion.
Appendix D

Extending FRASR

FRASR has several points at which it can be easily extended (see Section 4.1). This appendix describes at which points FRASR can be extended and what kind of tasks each of those extensions should implement.

D.1 Data source

One of the most obvious point of extension of FRASR, is the possibility to add support for data from software repositories which are not yet supported by FRASR. Section 9.1 describes a number of examples of such software repositories.

Each data source contains a number of parameters which the user can use to provide the settings of the data source. These can for instance be a URL, username, password, etcetera.

Furthermore, each data source contains a number of data fields. These fields can be used in the filter and the event bindings views in the GUI. Each data field has an associated type (which must implement the Comparable [Jav10a] interface of java.lang, in order to be able to apply the filter operators on the values of the type).

The component case has three variants (see Section 4.1.2). Each data source must indicate which variant should be used for that data source. Furthermore, each data source must indicate which data fields should be used for this variant.

In order to extract events from the data source, the data source must be able to convert the ‘raw’ data from the data source to events from the log model used in FRASR (see Section 4.1.3). This process should also handle the filtering of the data. A data source may provide a cache option. When such an option is available, the data source should handle the source from which the events should be fetched (the source, the cache, or a combination of the source and the cache).

Data sources may have special event bindings associated with them. These bindings must be separately defined, but should be associated by the data source. See Section D.3 for more information.

D.2 Case mapping

An other point at which FRASR can be extended, is the case mappings. At the moment, five different case mappings have been implemented, see Figure 4.3 in Section 4.1.2. Other case mappings, which might give an other view at the data in the analysis application, can be for example a case mapping in which the data source type (or data source category) is used as
the case instance. The point at which the domain model can be extended, is indicated by `<ID>Case`.

### D.3 Event binding

Event bindings define the way in which events are extracted from the data of the data sources. Event bindings can be specific for a data source, or applicable to all data sources. Figure 4.3 in Section 4.1.2 shows the points at which the domain model can be extended. The `<ID>EventBinding` represents a general purpose (or not data source specific) event binding. `DS_<DataSource>EventBinding` and `DS_<DataSource>ComponentEventBinding` represent data source specific event bindings.

`DS_<DataSource>EventBinding` represents a data source specific binding which defines its own events names. Examples for a specific bug tracker are: `Ticket-imported`, `Ticket-suspended`, `Ticket-created`, etcetera. `DS_<DataSource>ComponentEventBinding` represents a data source specific binding, which implements one of the three variants of component event bindings, which can be used to map data from some data fields of a data source to a predefined set of keywords for example.

### D.4 Export type

FRASR is an application which facilitates process mining. Therefore, it supports a number of export formats to which the data from the data sources (which satisfies the internal log model, see Figure 4.4 in Section 4.1.3) can be exported.

In order to use FRASR in combination with an analysis application which does not support one of the export formats supported by FRASR, FRASR can be extended by providing support for this specific export type.

Each export type should provide a file type as well as a description of the export type.

### D.5 Cache implementation

The current implementation of the cache stores all the data to files (see Section 5.1). An other option could be for example to send all the data to a web server, from which other FRASR users can easily download the data (assuming they use the same project files).

Each cache implementation in FRASR should implement methods to store, retrieve and delete data given a data source.