



Free/Libre and Open Source Software Metrics

Sponsored through Framework Programme Sixth (Call 5) by



The FLOSSMetrics Consortium consists of: Universidad Rey Juan Carlos, University of Maastrich, Wirtschaftsuniversitaet Wien, Aristotle University of Thessaloniki, Conecta s.r.l., Zea Partners and Philips Medical Systems PMS Nederland B.V.

Document Information

Version: 2.0
Date : Oct 11, 2008
revision: 1

Owning Partner: AUTH

Author(s):
 Ionannis Antoniadis
 Ioannis Samoladas
 Sulayman K. Sowe
 Gregorio Robles
 Stefan Koch
 Ksenia Fraczek
 Anis Hadzisalihovic
 Daniel Izquierdo-Cortazar

Reviewer(s):
 Stefan Koch

To:
 PUBLIC

Purpose of distribution:
Final Version

Printed on at

Status:

- Draft
- To be reviewed
- Proposal
- Final/Released

Confidentiality:

- Public - Intended for public use
- Restricted - Intended for FLOSSMETRICS consortium only
- Confidential - Intended for individual partner only

Deliverable ID: D1.1

Title:
 Study of Available Tools

License for distribution:

This work is licensed under a [Creative Commons Attribution-Share Alike 2.5 License](http://creativecommons.org/licenses/by-sa/2.5/).
 (The license can be found in <http://creativecommons.org/licenses/by-sa/2.5/>)



Study of Available Tools

Deliverable ID: D1.1

Page : 2 of 75

Version: 2.0

Date: Oct 11, 2008

Status : Final


Confid : Public

Deliverable: D1.1

Title: Study of Available Tools

Executive Summary:

This deliverable identifies, analyzes and evaluates all possible data sources on FLOSS projects with special emphasis on data available for the next deliverables. Especially, repositories of FLOSS are discussed, then the main information sources versioning systems, mailing lists and bug tracing systems are described.

	Study of Available Tools Deliverable ID: D1.1	Page : 3 of 75
		Version: 2.0
		Date: Oct 11, 2008
		Status : Final Confid : Public

CHANGE LOG

Ver.	Date	Author	Description
0.1	05/12/2006	Ionannis Antoniadis Ioannis Samoladas Sulayman K. Sowe Gregorio Robles Stefan Koch Ksenia Fraczek Anis Hadzisalihovic	Initial version
0.9	13/2/2007	Ionannis Antoniadis Ioannis Samoladas Sulayman K. Sowe Gregorio Robles Stefan Koch Ksenia Fraczek Anis Hadzisalihovic	Review Version
1.0	15/2/2007	Stefan Koch	Final Version
2.0	11/10/2008	Daniel Izquierdo-Cortazar	Matrices, tools and metrics added.

APPLICABLE DOCUMENT LIST

Ref.	Title, author, source, date, status	Deliverable Identification

Contents

1	Repositories of FLOSS and Repositories of Repositories	6
1.1	Repositories of FLOSS	6
1.2	Available Data in FLOSS Repositories	7
1.3	Trends in FLOSS Studies and RoRs	8
2	Versioning Systems	12
2.1	Introduction	12
2.2	Structure and Function	12
2.3	Preprocessing: retrieval and parsing	13
2.4	Data treatment and storage	17
2.5	CVS	18
2.6	Subversion	19
2.6.1	Subversion Repository Access	19
2.6.2	Comparison between Subversion and CVS	19
2.6.3	Subversion Logs	20
2.6.4	Subversion Repository Analysis	22
3	Communication Tools - Mailing Lists	23
3.1	Introduction	23
3.2	Activities in Mailing Lists	24
3.3	Mining mailing lists for FLOSS development and support	24
3.4	Knowledge Sharing Metrics	29
4	Bug-Tracking Systems	31
4.1	Introduction	31
4.2	Bug Tracking Systems	34
4.3	Bugzilla	35
4.3.1	General description	35
4.3.2	Features	35
4.3.3	Description	36
4.3.4	Options	37

4.3.5	Design	38
4.3.6	Architecture	41
4.3.7	Benefits	41
4.3.8	Usage	42
4.3.9	Data available	42
4.3.10	Studies & Retrieval Tools	46
4.4	Source Forge Tracker	47
4.4.1	Technology	49
4.4.2	Architecture	51
4.4.3	Tracker	51
4.5	GNATS	53
4.5.1	Architecture	54
4.5.2	GNATSweb	57
4.5.3	TkGnats	57
4.6	PHP Helpdesk	58
4.7	PHPBugTracker	59
4.8	Double Choco Latte	62
A Data Sources		69
B Tools and Metrics		75
B.1	Tools	75
B.2	Metrics	76

Chapter 1

Repositories of FLOSS and Repositories of Repositories

1.1 Repositories of FLOSS

Participants in FLOSS project rely on extensive peer collaboration through the Internet, using project's mailing lists, *de facto* versioning systems such as Concurrent Versions System (CVS) or Subversion (SVN), bug-tracking systems (BTS) and bug databases (e.g. Bugzilla), Internet Relay Chats (IRC), discussion forums, etc. These tools not only enable participants to collaborate in the software development process but also act as repositories to store the communication activities of the participants¹.

With the coming of FLOSS, various portals started to provide hosting services for FLOSS projects of all kinds and flavors. Among the largest and most popular these is SourceForge.net. Other portals such as FreshMeat.net and Savannah.gnu.org also continue to attract a lot of attention. These portals are hosts to small and large, successful and unsuccessful projects. Yet, many portals are also graveyards strewed with abandoned projects. The plethora of FLOSS projects or applications available throughout the Internet is an indication of a growing interest in F/OSS and the fact that an increasing number of skilled programmers are willing to transform their (tacit) knowledge and skills into tangible products [SIA06].

Many researchers use data obtained from direct access to repositories such as SourceForge.net [BR05, XGCM05], for example. Despite having firsthand access to the data source, harvesting data or crawling SourceForge.net could be a daunting task for researchers [HC04]. Alternatively, researchers may use subsidiary meta-data provided in a repository of repos-

¹A list of categories and types of repositories can be found at appendix A

itories or RoRs such as FLOSSmole [JMC06, HC04]. FLOSSmole (<http://ossmole.sourceforge.net/>) may be described as a repository of repositories (RoRs) or meta-repository of raw data of F/OSS projects hosted at various portals such as FreshMeat, SourceForge.net, Rubyforge.org, and Objectweb.org. The opportunities researchers have to obtain data directly from repositories or reuse data dumps already used by other researchers have, to a large extent, changed the way we obtain data for quality FLOSS research. The FLOSSMole interface is shown Figure 1.



Figure 1.1: FLOSSMole interface

1.2 Available Data in FLOSS Repositories

The data available in FLOSS repositories stems for two major sources:

1. Tools offered: Web sites which host F/OSS projects also provide each project with repositories or tools to enable the collaborative software development process to proceed. The largest web site which provides these services to projects and has generated a lot of research interest is SourceForge.net. The tools offered include versioning systems, mailing lists or bug tracking systems. These tools are described in their own sections, as they are a general concept which can be used by any project.

2. Metadata: Most repositories also maintain metadata on the projects hosted. This can range from simple measures like the number of registered users, to number of downloads, activity metrics or development status. These metrics are either reported by the project's participants, or computed from other available data (like the number of downloads). These data can also be accessed and used, but are mostly repository-specific. Regarding the example of Sourceforge.net, data from this site has been used to study many aspects of F/OSS; the geographical location of F/OSS developers [RGB06], topological analysis of developer communities [XGCM05], knowledge collaboration across projects and mailing lists [SIA06], patterns of software development [SDD05], percentage distribution of projects, identification of projects with a certain number of listed developers and bugs [HC04], etc.

1.3 Trends in FLOSS Studies and RoRs

Compared to traditional research practices under proprietary software, F/OSS development provides researchers with an unprecedented abundance of easily accessible data for research and analysis. A huge amount of data is available to study community participation in F/OSS projects [HM05, Mas05] and developer and user involvement in projects mailing list [SIA06, KSL03].

Although the traditional way of obtaining data for most of these kinds of research is spidering or crawling of SourceForge using scripts (Perl, Python), participant observation, personal interviews or a combination of more than one approach have been used. Instead of direct access to the SourceForge repository, researchers may benefit from reusing data obtained from SourceForge. The University of Notre Dame maintains a data dump from SourceForge [BR05] and other researchers may request and reuse the data in their studies [RGB06]. In another study [VTG⁺06], reused data from [HC04] to study the self-organizing patterns in wasp and F/OSS communities.

Research Difficulties

It is becoming increasingly evident that collecting and analyzing F/OSS data has become a problem of abundance and reliability in terms of storage, sharing, aggregation, and filtering [CO006]. Some of the problems researchers may face in obtaining and using data in their research can be summarize thus:

- **Convergence of data:** There is no standardized way of defining or a naming convention for variables in a repository. This may pose prob-

lems for researchers when it comes to harmonizing data across different repositories.

- **Without Notice!**: The data structure of a repository is held in a back-tier (database). And because many researchers just interact with the front-end of the repository, researchers can face a daunting task when a site or a repository maintainer changes the structure of the data or schema [HC04].
- **Confidentiality**: Due to the sensitive nature of some aspects of the data (e.g. private emails), some projects might be reluctant to release some of their data at a time when the researcher actually needs it.
- **A friend of a friend (*FOAF*)**: A researcher not having direct access to the data he needs for his research may send a request to the project, either through mailing lists or to the repository maintainer. Experience shows that sometimes processing such a request is like waiting for rain in the desert. In this case, knowing someone who has obtained the data the research wants or knowing some members of the core team helps.

Possible Solutions

These difficulties significantly impede F/OSS research. As a result, many researchers see the need for the establishment and use of **Repositories of Repositories** or *RoRs*. The whole concept of RoRs is an attempt to pull data from many and varied repositories and bring that data under one umbrella so that researchers can have easy access to data, reports, tools, and scripts used in F/OSS research. As [HC04] noted in their schematic analysis, current F/OSS research is non-cyclical and non-collaborative. It is 'one-way traffic'. Once researchers obtain, analyze and publish their data, the products of their research is never put back to the community from which they obtained their data. Existing structures of software repositories do little to ameliorate this situation. The aim of RoRs should be to close this loop by encouraging researchers to contribute their data and any scripts and tools they used in their research to the RoRs, from which they obtained the original data. Our view of how the RoRs concept should work is illustrated in Figure 2. From the diagram, note that there is a continuous feedback between the research community and the RoRs. The first kind of RoRs available to researchers is FLOSSmole. For a detailed description of the purpose, design, and requirements of Flossmole, see [HC04]. Another RoRs in progress is the EU funded FLOSSMetrics (<http://www.flossmetrics.org/>) project. FLOSSMetrics or Free/Libre Open Source Software Metrics project aims to

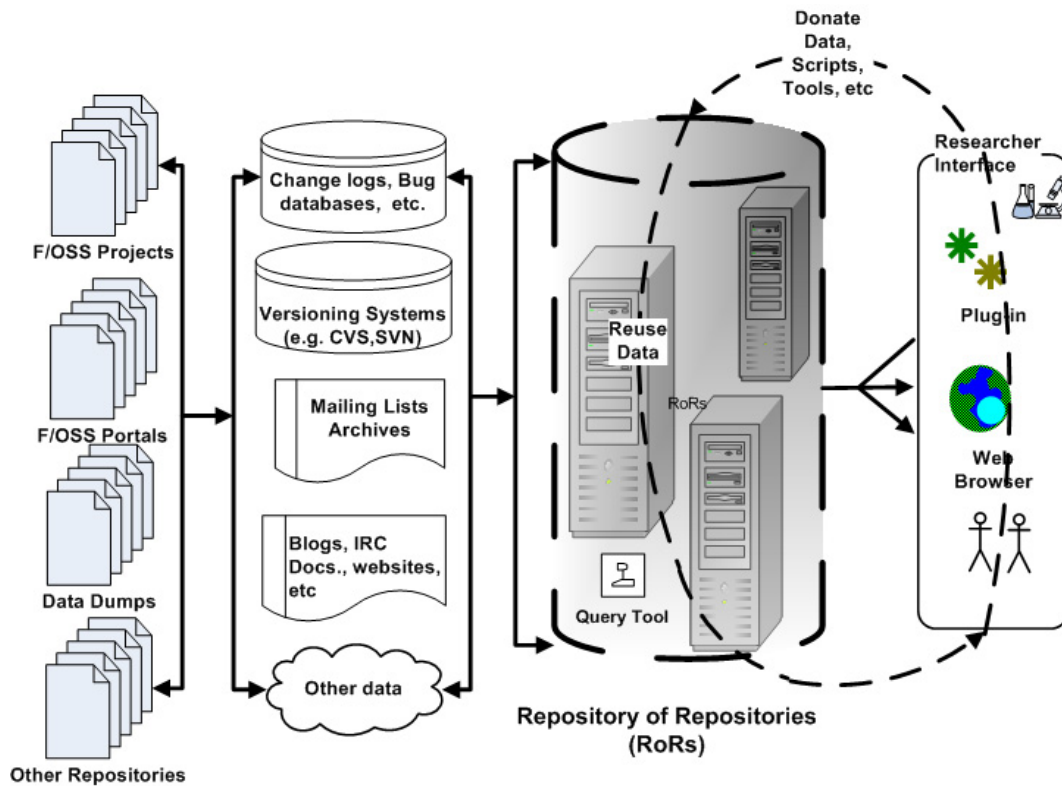


Figure 1.2: Conceptual Framework of RoRs

construct, publish and analyze a large scale database with information and metrics about F/OSS development. Using existing methodologies and tools already developed the project will house data coming from several thousands of software projects. The project will also provide a public platform for validation and industrial exploitation of results. Some of the targets of the project are summarized.

- Identify, evaluate sources of data, and develop a database structure.
- Build and maintain an updated empirical database.
- Disseminate the results, including data, methods and software.

The roadmap of the project is as shown in Figure 3.

The project will work with other projects such as FLOSSmole, the Software Quality Observatory for Open Source Software or SQO-OSS (<http://www.sqo-oss.eu/>) and QUALity of Open Source Software or QUALOSS

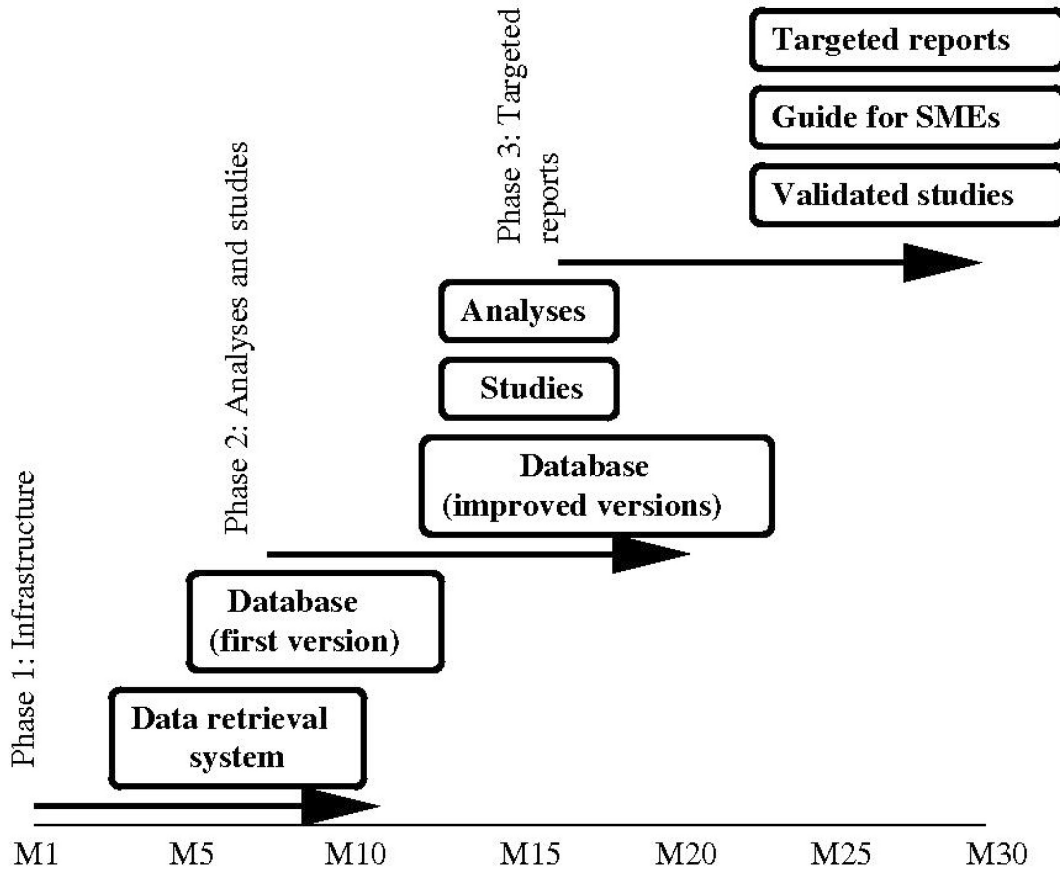


Figure 1.3: FLOSSMetrics Project's Roadmap. M=months

(<http://www.qualoss.org/>). However, the targeted studies of FLOSSMetrics are clearly different from those in QUALOSS and SQOOSS. Thus, it is becoming increasingly feasible to use data from RoRs for quality F/OSS research.

Chapter 2

Versioning Systems

2.1 Introduction

Versioning systems are used to manage file versions in a software development project. Thus they allow to track changes and past states of a software project. So, obtaining the current and any past state of the code is made possible by the use of a versioning system. This allows to make source code analysis in a longitudinal manner and to extract facts on the evolution of a software project.

But beyond this, versioning systems store a set of meta-data of the changes. These meta-data can be tracked and analyzed. This information is usually related to the interactions that occur among developers and the versioning systems. In general the information is only related to actions that comprehend write access while reading (downloading the sources) or obtaining other information (diffs, among others) cannot be tracked in that way.

In the following sections we are going to present a generalised framework for versioning systems (repositories) analysis. In our presentation we are going to use CVS as an example, but the same methodology is true for other versioning systems.

2.2 Structure and Function

By means of studying a CVS repository, any interaction - also called commit - a commiter¹ does with the central versioning system repository is logged with following data associated: commiter name, date, file, revision number,

¹A commiter is a person who has write access to the repository and does a commit - an interaction - with it at a given time.

lines added, lines removed and an explanatory comment introduced by the commiter. There is some file-specific information that can also be extracted, as for instance if the file has been removed². On the other hand, the human-inserted comment can also be parsed in order to see if the commit corresponds to an external contribution or even to an automated script.

Basically all tools that analyze CVS repositories consist of three main steps, preprocessing, storage and post-processing, but they can be subdivided into several more.

2.3 Preprocessing: retrieval and parsing

Preprocessing includes downloading the sources from the CVS repository of the project in study. Afterwards, aggregated modules³ have to be removed to avoid counting commits several times. Once this is done, the logs are retrieved and parsed to transform the information contained in log format into a more structured format (SQL for databases or XML for data exchange).

Besides the information for every commit, other data obtained from the parsing requires some attention. Although committers seldom change their username, sometimes this happens, so the various usernames have to be merged into a unique one. For instance, in the KDE project committers usually get a CVS account prior to a *kde.org* e-mail address. If a developer is afterwards assigned an e-mail address the username of e-mail and CVS have to be identical for organizational and practical reasons. If the username in the e-mail address is different from the CVS username, CVSanaly syncs with the former one and the actions done with both usernames are considered as done by a unique developer.

The following is a CVS log excerpt for the AUTHORS file of the KDevelop project⁴. It gives the last 6 revisions (from revision 1.45 to 1.49) done during the last months of the year 2003 until mid-2004.

[...]

RCS file: /mirrors/kde//kdevelop/AUTHORS,v
Working file: /mirrors/kde//kdevelop/AUTHORS

²In a versioning system there is actually no file deletion. In the case of CVS, files that are not required anymore are stored in the Attic and may be called back anytime in future.

³ Aggregated modules are modules that are shared between other modules. Such modules generally include system-wide administration and scripts. This information is kept in the CVSROOT/modules file.

⁴KDevelop is an IDE (Integrated Development Environment) for KDE. More information can be obtained from <http://kdevelop.org/>.

```

head: 1.49
branch:
locks: strict
access list:
keyword substitution: kv
total revisions: 103;   selected revisions: 103
description:
-----
revision 1.49
date: 2004/06/21 18:57:13;  author: rgruber;  state: Exp;  lines: +4 -0
Added self
-----
revision 1.48
date: 2004/02/24 14:42:59;  author: dagerbo;  state: Exp;  lines: +5 -1
Added self :)
-----
revision 1.47
date: 2004/02/15 22:40:33;  author: aclu;  state: Exp;  lines: +3 -3
Some more credits update.
-----
revision 1.46
date: 2004/02/15 22:02:33;  author: geiseri;  state: Exp;  lines: +6 -0
I guess I need to accept the blame for these things...
-----
revision 1.45
date: 2003/11/01 11:47:30;  author: dhaumann;  state: Exp;  lines: +4 -1
branches: 1.45.2;
KTabWidget -> KDevTabWidget,
added author.

[...]

```

While being parsed each file can be matched for its type. This can be done by several means: the easiest way is to look at its extension, but also common file names or the content can be inspected. This separation may help in the identification of different contributor groups that work on the software, so besides source code files the following file types may also be considered: documentation (including web pages), images, translation (generally internationalization and localization), user interface and sound files. If we only consider the information that appears in the CVS logs, then the possibility of looking at the content of the files does not exist, so only file

names and file extensions can be taken into account.

CVS also has some peculiarities when introducing contents for the first time (this action is called initial check-in). The initial version (with version number 1.1.1.1) is not considered in our computation as it is the same as the second one (which has version number 1.1). The number of aggregated and removed lines in CVS are computed from this initial version. This means that the first commit (the initial check-in) logs as if 0 lines were added. This does not correspond to reality. In order to obtain the actual number of LOCs in the first version we can count the LOCs by means of the UNIX *wc* tool of the latest version, subtracting the added lines and adding the removed lines of all the other commits.

Comments attached to commits are usually forwarded to a mailing list so that developers keep track of the latest changes in CVS. Some projects have established some conventions so that certain commits do not produce a message to the mailing list. This happens when those commits are supposed to not require any notification to the rest of the development team. A good example of the pertinent use of *silent* commits comes from the existence of bots that do several tasks automatically.

In any case, such conventions are not limited to non-human bots, as human committers usually may also use them. In a large community, we can argue that *silent* commits can be considered as not contributory (i.e. changes to the head of the files, for instance a change in the license or the year in the copyright notice, or moving many files from one location to another). Therefore, a flag for such commits can be set in order to compute them separately or leave them out completely in our analysis.

For instance, the developers of the KDE project mark such commits with the comment *CVS_SILENT* as it can be seen from following log excerpt extracted from the `kdevelop_scripting.desktop` file of the KDevelop CVS module. In this case it is due to a change to a *desktop* file, a file type that is related to the user interface. Being this change not considered interesting for other developers to know about, the author of this commit decided to make this commit *silently*.

[...]

```
RCS file: /mirrors/kde//kdevelop/kdevelop_scripting.desktop,v
Working file: /mirrors/kde//kdevelop/kdevelop_scripting.desktop
head: 1.24
branch:
locks: strict
access list:
```

```

keyword substitution: kv
total revisions: 30;    selected revisions: 30
description:
-----
revision 1.24
date: 2005/03/28 03:29:25;  author: scripty;  state: Exp;  lines: +2 -2
CVS_SILENT made messages (.desktop file)
-----
revision 1.23
date: 2005/03/27 04:05:51;  author: scripty;  state: Exp;  lines: +4 -0
CVS_SILENT made messages (.desktop file)
-----

```

[...]

Write access to the versioning system is not given to anyone. Usually this privilege is granted only to those contributors who have reached a compromise with the project and the project's goals. But external contributions -commonly called patches, that may contain bug fixes as well as implementation of new functionality- from people outside the ones who have write access (committers) are always welcome.

It is a widely accepted practice to mark an external contribution with an authorship attribution when committing it. Thus, certain heuristics can be constructed to find and mark commits due to such contributions. The heuristics that can be set up are based on the appearance of two circumstances: keywords such as patch (or patches in its plural form) together with a preposition (from, by, of, and other) or an e-mail address or an indication that the code had been attached to a bug fix in the bug-tracking system. The regular expressions that are widely in use are following:

- [1] patch(es)?\s?.* from
- [2] patch(es)?\s?.* by
- [3] patch(es)?\s?.*@
- [4] @.* patch(es)?
- [5] 's.* patch(es)?
- [6] s' .* patch(es)?
- [7] patch(es)? of
- [8] <.* [Aa][Tt] .*>
- [9] attached to #

As an example, the following slightly modified excerpt taken from the kdevelop.m4.in file from the KDevelop module in the KDE CVS repository

shows a patch applied by a commiter with username dymo that was submitted originally by Willem Boschman:

```
[...]  
-----  
revision 1.39  
date: 2004/06/11 17:07:57; author: dymo; state: Exp; lines: +3 -3  
Applied patch from Willem Boschman -  
fix builddir != srcdir configuration problem.  
-----  
[...]
```

All these efforts have in common that they perform text-based analysis of the comments attached by committers to the changes they perform. The range of possibilities in this sense is very ample. For instance, Mockus et al. tried to identify the reasons for changes (classifying changes as adaptative, perfective or corrective) in the software using text-analysis techniques [MV00].

2.4 Data treatment and storage

Once the logs have been parsed and transformed into a more structured format, some summarizing and database optimization information is computed and data is stored into a database.

Usually the output of the previous parsing consists of a single database table with an entry per commit. This means that information is stored in a raw form, containing the table possibly millions of entries depending on the size and age of a project⁵. Information is hence in a raw format and in an inconvenient way if we consider getting statistical information for developers and projects from it.

A first step in this direction is to make use of normalization techniques for the data. In this sense, committers are assigned a unique numerical identification and if further granularity is needed, procedures have been implemented to do the same at the directory and file level. For the sake of optimization this can be introduced during the parsing phase so additional queries have not to be performed. The next step is to gather statistical information on both committers and modules. These additional tables will give detail on the

⁵For instance, as of April 2005, over 7 million commits to the CVS of the KDE project have been computed. The number of commits for other projects such as GNOME or Apache is also in the order of magnitude of millions of commits.

interactions by contributors or to modules, which is one of the most frequent information that is asked.

Additional information that makes longitudinal analysis possible is the evolution of contributions by developers and to modules. Hence, the same statistical queries that have been obtained for committers and modules for the summarizing tables can be obtained in a monthly or weekly basis since the date the repository was set up.

On the other hand, unfortunately CVS does not keep track of which files have been committed at the same time. The absence of this concept in CVS may bring some distortion into our analysis. There are several sliding window algorithm proposed by German [Ger04] and Zimmermann et al. [ZW04] that identify atomic commits (also known as *modification records* or transactions) by grouping commits from the CVS logs that have been done (almost) simultaneously. This algorithm considers that commits performed by the same commiter in a given time interval (usually in the range of seconds to minutes) can be considered as an atomic commit. If the time window is fixed, the amount of time that is considered from the first commit to the last one is a constant value. For a sliding time window, the time interval is not constant; the time window is restarted for every new commit that belongs to the same transaction until no new commit occurs in the (new) time slot [ZW04].

The post-process may be composed of routines that interact with the database, analyze statistically its information, compute several inequality and concentration indexes and generate graphs for the evolution over time for a couple of interesting parameters (commits, committers, LOCs...).

For instance, results of a CVS analysis can be obtained through a publicly accessible web interface that allows easy inspection of the whole repository (general results), a single module or by committers. Therefore results are available for remote analysis and interpretation by project participants and other stakeholders.

2.5 CVS

CVS, Concurrent Versioning System, has historically been the most used versioning system used in libre software projects. For instance, more than 10,000 projects hosted at SourceForge are reported to use it in 2003 [HS03], and the 11 largest projects in Debian 2.2 [GBPdlHQ⁺01] use versioning systems (all of them CVS except for Linux that at that time used Bitkeeper, a proprietary versioning solution⁶). As CVS has been used as an example in

⁶As of June 2005, the Linux project moved to another versioning tool developed by the Linux developers.

the treatment above, no additional analysis of this tool is given here.

2.6 Subversion

Subversion⁷ is an open source application used for revision control. Subversion is designed specifically to be a modern replacement for CVS. Many open source projects are now moving to subversion to benefit from its modern design and features.

2.6.1 Subversion Repository Access

Subversion repositories can be accessed by the following means:

- Local file system or network file system, accessed by client directly.
- WebDAVDeltaV (over `http` or `https`) using the `mod_dav_svn` module for Apache 2.
- Custom `svn` protocol, either plain text or over SSH.

All three means can be used for all subversion operations.

2.6.2 Comparison between Subversion and CVS

Although the majority of open source projects use CVS as their primary versioning control system, there is a continuous switch to subversion. Major repository hosting providers, such as Sourceforge.net, are replacing their versioning systems with subversion. The table below presents a comparison between CVS and Subversion.

	CVS	Subversion
Atomic Commits	Commits are not atomic	Commits are atomic
Rename support	No	Yes
Copy of files to a different location	No	Yes
Remote Repository Replication	No	Yes (indirectly)
Repository permissions	Limited	

The command set that subversion uses is very similar to that of CVS, for example in order to update a local repository, with CVS the command is `cv`**s** `update` while with subversion it is `sv`**n** `update`.

⁷<http://subversion.tigris.org>

2.6.3 Subversion Logs

With the command `svn log` someone can see the log messages for all files and directories inside of (and including) the current working directory of your working copy. With the option `-v` (`svn log -v`) `svn` will also print all affected paths with each log message. For example:

```
r26 | robflynn | 2000-03-23 14:44:52 +0200 (, 23 2000) | 2 lines
Changed paths:
  M /trunk/gaim/src/util.c
```

One of the previous patches didnt go in correctly. Its fixed now.

```
r25 | robflynn | 2000-03-23 13:12:23 +0200 (, 23 2000) | 2 lines
Changed paths:
  M /trunk/gaim/src/gnome_applet_mgr.c
```

Another patch from Eric.

```
r24 | robflynn | 2000-03-23 12:18:02 +0200 (, 23 2000) | 2 lines
Changed paths:
  M /trunk/gaim/ChangeLog
  M /trunk/gaim/src/gaimrc.c
```

Fixed problem with away messages.

```
r23 | robflynn | 2000-03-23 11:53:31 +0200 (, 23 2000) | 2 lines
Changed paths:
  M /trunk/gaim/ChangeLog
  M /trunk/gaim/src/server.c
```

Small fix to the lagometer

Apart from the text format, `svn` offers an option to extract logs in xml format, allowing for easier analysis (`svn --xml -v log`). For example the previous entries are transformed into the following xml format.

```
<logentry
```

```

    revision="25">
<author>robflynn</author>
<date>2000-03-23T11:12:23.000000Z</date>
<paths>
<path
  action="M">/trunk/gaim/src/gnome_applet_mgr.c</path>
</paths>
<msg>Another patch from Eric.
</msg>
</logentry>
<logentry
  revision="24">
<author>robflynn</author>
<date>2000-03-23T10:18:02.000000Z</date>
<paths>
<path
  action="M">/trunk/gaim/ChangeLog</path>
<path
  action="M">/trunk/gaim/src/gaimrc.c</path>
</paths>
<msg>Fixed problem with away messages.
</msg>
</logentry>
<logentry
  revision="23">
<author>robflynn</author>
<date>2000-03-23T09:53:31.000000Z</date>
<paths>
<path
  action="M">/trunk/gaim/ChangeLog</path>
<path
  action="M">/trunk/gaim/src/server.c</path>
</paths>
<msg>Small fix to the lagometer
</msg>
</logentry>

```

The `svn blame` command shows author and revision information in-line for the specified files or URLs. Each line of text is annotated at the beginning with the author (username) and the revision number for the last change to that line.

```

8749 faceprint
9474 lschiere      if (imhtml->edit.link)
9474 lschiere      gtk_imhtml_toggle_link(imhtml, NULL);
9474 lschiere
9451 lschiere
gtk_text_buffer_remove_all_tags(imhtml->text_buffer, iter, iter);
9430 seanegan
8749 faceprint }
8749 faceprint
8749 faceprint char *gtk_imhtml_get_markup(GtkIMHtml *imhtml)
8749 faceprint {
8749 faceprint      GtkTextIter start, end;
8749 faceprint
8749 faceprint
gtk_text_buffer_get_start_iter(imhtml->text_buffer, &start);
8749 faceprint
gtk_text_buffer_get_end_iter(imhtml->text_buffer, &end);
8749 faceprint      return gtk_imhtml_get_markup_range(imhtml,
&start, &end);
8749 faceprint }
8749 faceprint

```

2.6.4 Subversion Repository Analysis

The structure of a subversion analysis tool is not different from that of a CVS analysis tool. Both tools use the same methodology on keeping track of changes, they differ in some technicalities with the versioning mechanism. So, the way we analyse a subversion repository and the methodology we use is the same as that of the CVS case. Of course parts such as retrieval and parsing are different in the command they use, but the rest are the same. All algorithms and considerations that are taken into account for the CVS analysis are applied to the analysis of subversion repositories.

Chapter 3

Communication Tools - Mailing Lists

3.1 Introduction

The FLOSS windfall is such that there is increased motivation to understand how software developers and users communicate and the overall nature of community participation in FLOSS projects. Substantial research is proceeding with focus on software repositories such as mailing lists to study developer communities with the ultimate aim to increase our understanding of core software development activities. Mundane project activities which are not explicit in most developer lists have also received attention [SIA06], [LvH03]. Many researchers focus on mailing lists in conjunction with other software repositories [KSL03, Gho04, LvH03, HK05, RBM03]. These studies provided great insight into the collaborative software development process that characterizes FLOSS. Community studies in mailing lists are important because on one hand, one major technical infrastructure FLOSS projects require is mailing lists. On the other hand, FLOSS projects are symbiotic cognitive systems where ongoing interactions among project participants generate valuable software knowledge - a collection of shared and publicly reusable knowledge - that is worth archiving [SIA06]. One form of knowledge repository where archiving of public knowledge takes place is the project's mailing list. Some other source code repositories such as bug-tracking and versioning systems are available to facilitate the software development and coordination process.

3.2 Activities in Mailing Lists

Lists are active and complex living repositories of public discussions among FLOSS participants on issues relating to project development and software use. They contain what [Ger03] called 'software trails' - pieces left behind by the contributors of a software project and are very important in educating future developers and non-developers [SIA06] on the characteristics and evolution of the project and software. Generally, a project will host many lists, each addressing a specifically area of need. For example, software developers will consult developer lists, participants needing help on documentation will seek links from lists associated with project documentation, beginners or newbies will confer with mentors' lists, etc. Fundamentally, two forms of activities are addressed in lists;

- **Core activities** typified by developing, debugging, and improving software. Developer mailing lists are usually the avenues for such activities.
- **Mundane activities** [MFH02, LvH03, KSL03]. Documentation, testing, localization, and field support exemplifies these activities and they take place predominately in non-developer lists [SIA06].

However, expert software developers, project and package maintainers take part in mundane activities in non-developer mailing lists. They interact with participants and help answer questions others posted. Sometimes they encounter useful issues which help them to further plan and improve code or overall software quality and functionality. In addition, although mundane activities display a low level of innovativeness, they are fundamental for the adoption of FLOSS.

3.3 Mining mailing lists for FLOSS development and support

Compared to the traditional way of developing proprietary software, FLOSS development has provided researchers with an unprecedented abundance of easily accessible data for research and analysis. It is now possible for researchers to obtain large sets of data for analysis or to carry out what [Gho04] referred to as 'Internet archaeology' in FLOSS development. However, [CO006] remarked that collecting and analyzing FLOSS data has become a problem of abundance and reliability in terms of storage, sharing, aggregation, and filtering of the data. FLOSS projects employ different kinds of

repositories for software development and collaboration. From these repositories community activities can be analyzed and studied. The figure below shows a methodology by which community participation in mailing lists may be studied. The methodology shows FLOSS project selection, choice of software repository and lists to analyze, data extraction schema, and data cleaning procedure used to extract results for analyzing community participation in developer and non-developer mailing lists.

Mailing lists participants interact by exchanging email messages. A participant posts a message to a list and may get a reply from another participant. This kind of interaction represents a cycle where posters are continuously internalizing and externalizing knowledge into the mailing lists. In any project's mailing list, these posters could assume the role of knowledge seekers and/or knowledge providers [SIA06]. The posting and replying activities of the participants are two variables that can be compared, measured and quantified. The affiliation an individual participation has with others as a result of the email messages they exchange within the same list or across lists in different projects could be mapped and visualized using Social Network Analyzes (SNA). For the construction of such an affiliation network or 'mailing list network' see [SIA06], pp. 130-131.

Mailing lists participants' may assume one or both of the following roles [SSA07]:

1. As a Knowledge Provider. A knowledge provider in FLOSS projects as an expert software developer who helps project participants on various issues related to software development and use.
2. As a Knowledge Seeker. Any list participant who seeks assistance on issues related to software development and use (e.g. how to compile code, run an application, configuration details, resolve package dependencies, documentation, etc) can be described as a knowledge seeker.

In FLOSS projects, developers are themselves users of the software. Most mailing lists are open to all participants so that users can ask questions, and developers can post patches for others to review. Sometimes a software developer or module maintainer may assume the role of a knowledge seeker by posting to lists, asking questions relating to software configuration, package dependency issues, bugs, etc. At the same time, an ordinary software user may assume the role of a knowledge provider by answering questions others ask in the lists. Roles are not assigned in FLOSS projects, almost every activity is voluntary. Depending on ones expertise, anyone can assume any role in the project. Thus, the distinction between a knowledge seeker and a

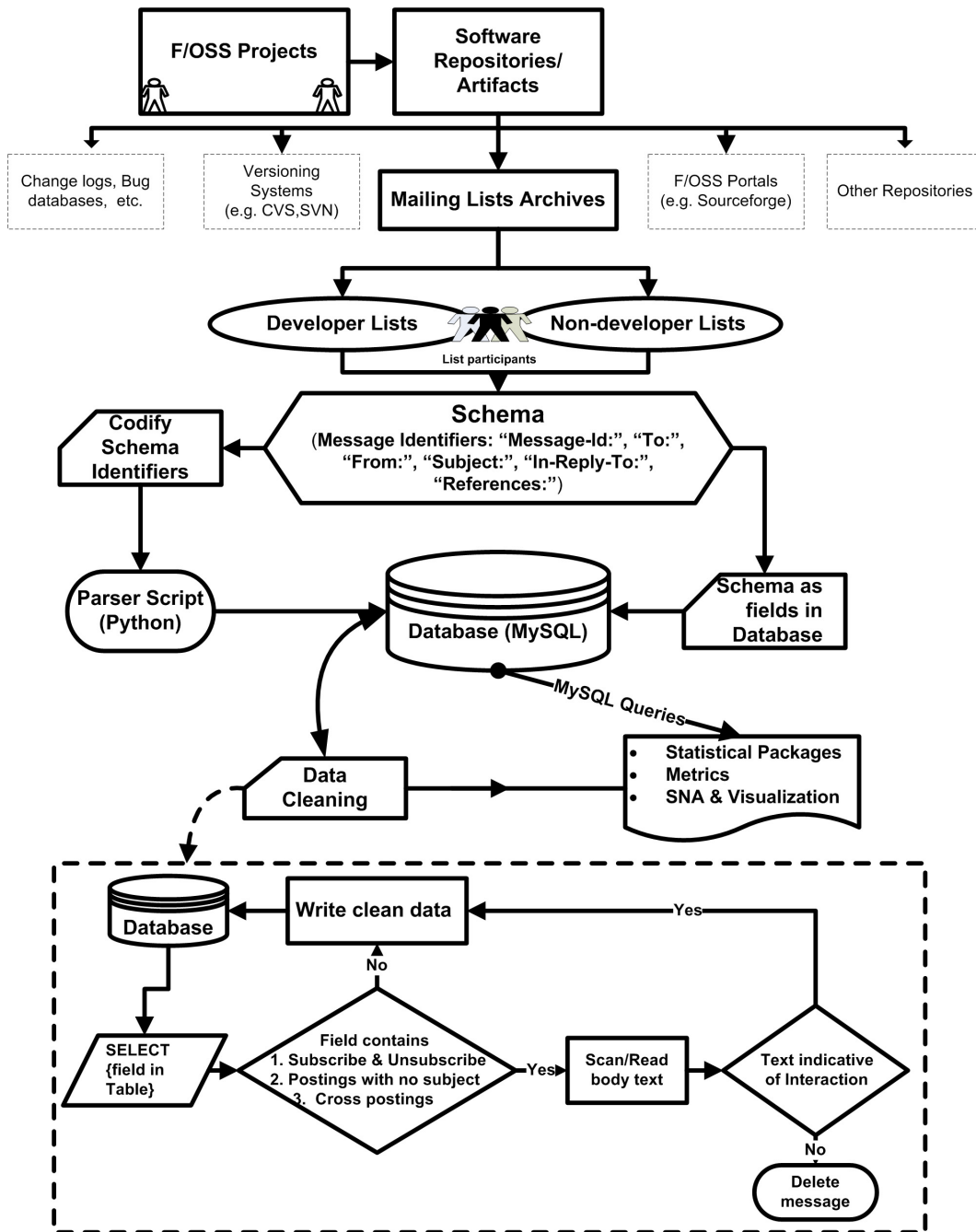


Figure 3.1: Methodological Outline to Extract Data from Mailing Lists Archives. Modified from [SIA06] (p.1027).

knowledge provider depends only on the unanticipated role of the individual at a particular moment in time.

Knowledge sharing between knowledge seekers and knowledge providers in FLOSS projects a synergistic process - "you get more out than you put in." If a mailing list participant shares his ideas or a way of installing or configuring particular software with another person, then just the act of putting his idea into words will help him shape and improve that idea. If he enters into a dialogue with the other mailing list participants, then he may benefit from their knowledge, from their unique way of doing things and improve his ideas further. Each list participant enters into a 'conversation' with other participants in the list. When two or more participants exchange email messages, they are said to share their knowledge. Knowledge sharing in FLOSS projects is all about helping each other and collaboration. The FLOSS development context is a symbiotic cognitive system, where the community learns from its participants, and each individual learns from the community ([SKS05], p.301). The benefit derived from knowledge sharing is that participants learn from each other ([SKS05], p.298), and the result of their interaction is archived in the project's mailing from which subsequent participants can learn.

Ongoing interactions between project participants are a means of acquiring valuable software knowledge that is worth archiving. Mailing lists play a vital role in connecting knowledge seekers who are searching for knowledge with knowledge providers who already possess this knowledge. Through mailing lists, software users can ask questions and get answers. Software developers can discuss code development; package maintainers can disseminate product updates, get feedbacks, and discuss bugs and software dependencies. The Knowledge Sharing Model (KSM) in Figure 1 shows how mailing list participants share their knowledge by exchanging one or more email messages. Their knowledge and concepts are archived into the project's mailing list or Knowledge Base [SKS05]. This is a collection of shared and publicly available artifact known as reusable or public knowledge. Other repositories include Concurrent Versions Systems (CVS), Frequently Asked Questions (FAQs), project web sites and bug databases, etc. Knowledge seekers and/or knowledge providers transfer their knowledge, expertise, or know-how to the mailing list by means of a process called externalization. The process of acquiring knowledge from the mailing list and filtering of that knowledge to provide greater relevance to the acquirer is called internalization [SSA07].

As directions of the arrows in Figure 1 show, a potential knowledge seeker composes a message by posting or externalizing (**A**) it to the list. A knowledge provider may internalize (**B**) that knowledge and reply to the post (**C**). For example, a potential knowledge seeker confronts an unfamiliar concept

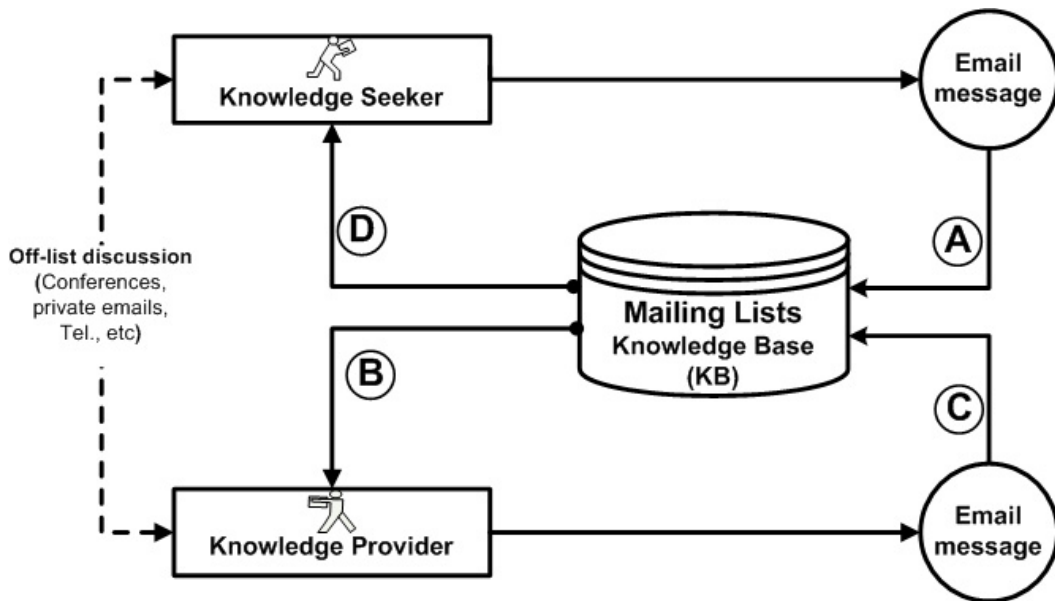


Figure 3.2: Knowledge Sharing Model.

(or bug) in the use of an application (e.g. OpenOffice) and decides to seek help from the project's mailing list. There are two ways to look at this scenario in the KSM:

- **If the concept has been encountered before**, it will be captured and stored in the knowledge base in the form of threaded discussions, which represents software knowledge resulting from interaction between list participants. This knowledge is externalized into the project's mailing list, indexed and archived, for subsequent knowledge seekers and knowledge providers to utilize by internalization. The knowledge seeker then directly consults the list.
 1. Knowledge seeker internalizes knowledge from knowledge base: $(KB) \implies (D)$.
According to this model, software experts or knowledge providers are also continuously browsing the mailing list to seek knowledge.
 2. Knowledge provider internalizes knowledge from knowledge base: $(KB) \implies (B)$.
The source of knowledge in both cases is the mailing list.
- **If the concept has not been encountered**, the problem solving part

of the model comes into play; the knowledge seeker’s problem has not been addressed in the mailing list before. It defines routes to be taken if a proposed solution fails, and how to respond if necessary information is absent [SSA07]. The knowledge seeker identifies the appropriate list, post his question, and exchange ideas with list participants.

1. Knowledge seeker posts his question to the list: $(\mathbf{A}) \implies (\mathbf{KB})$.
2. Knowledge provider may browse and reply to the list for every participant to benefit: $(\mathbf{C}) \implies (\mathbf{KB})$.
3. Knowledge seeker internalizes knowledge from knowledge base: $(\mathbf{KB}) \implies (\mathbf{D})$.

Alternatively, he may know a participant with expertise in the area he is interested in and exchange direct emails with that person (shown in dotted lines), with or without copies being posted to the list.

3.4 Knowledge Sharing Metrics

In the FLOSS context, participant must interact with the entities (e.g. websites, forums, to-do lists, etc) in which explicit knowledge is contained to have understanding of what the project is all about. However, it does not follow that another project participant can comprehend and correctly value the knowledge due to differences in programming capabilities or experience in the project [SSA07]. Thus, the process of measuring knowledge is complicated by its intangible nature, especially measuring tacit knowledge. But when tacit knowledge is transformed into explicit knowledge through socialization or interaction [49] and shared by members of an organization or project, an attempt can be made to measure how much knowledge is being shared. Koh and Kim, [KK04], suggested a way we can quantify the level of knowledge sharing in virtual communities. Similarly, we can provide quantifiable measures of knowledge sharing activities in FLOSS projects’ mailing lists by analyzing substantial email exchanges between list participants. We accomplish this in our KSM model by

1. counting the total number of posts externalized to the list. That is, email messages potential knowledge seekers posted. This quantity can be represented by the *nposts* variable,
2. counting the total number of replies made by potential knowledge providers to questions posted to the lists. This value quantity can be represented by the *nreplies* variable.

These two values will provide us a simple measure of the level of knowledge sharing in the lists.

Chapter 4

Bug-Tracking Systems

4.1 Introduction

It's hard to tell precisely how well the error-reporting system is working, but this seems to be a bug weapon that has landed a permanent spot in Microsoft's arsenal. [Deu04]

Bug – Tracking is a method which shows us whether the process is erroneous or not. It is found in different scientific methods like operational management, operational research, business intelligence and software. It is used in different sectors like:

- Software
 - development
 - maintenance
 - evolution
 - requirements analysis
 - testing
- Project Management
 - employees processing
 - workgroups
 - implementation
 - using the model Computation-, Artificial- und Knowledge-Intelligence
 - * theoretical terms and definitions in the model checking logic

- * data – warehouse, data – mining
- * neural networks

Definition 1 Bug tracking is an application for helping the software programmers and normal users in searching bugs.

After using bug tracking the system comes with clearly, precise, identifiable and establishable outputs without bugs.

Definition 2 Bug is a name for a program error e.g. it is an unknown intruder.

Historically seen, the first bug found was on paper.

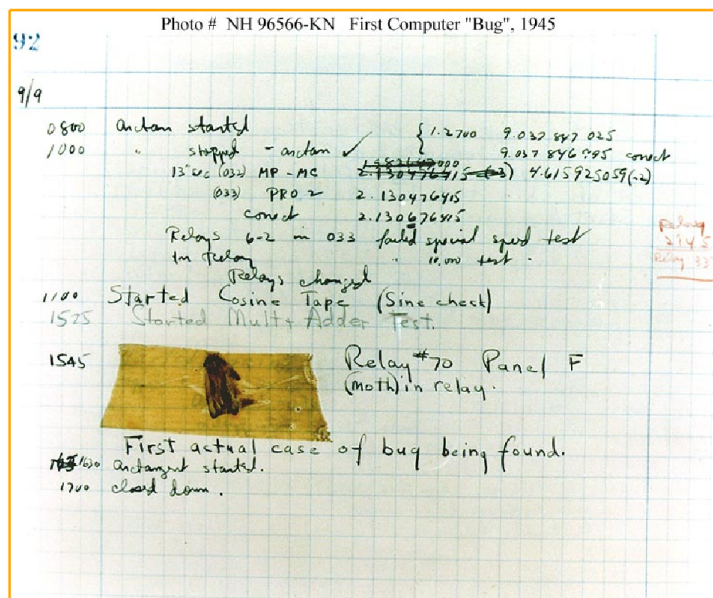


Figure 4.1: First Computer Bug [Wik06a]

Definition 3 Bug-tracking helps the software developers in knowing what is the error, resolving it, and learning from it. [IEE06]

Definition 4 Bug-tracking systems aren't a panacea for what ails software—they're useful for collecting aggregate data on how often certain kinds of faults occur. [Deu04]

The bug tracking appearance in the business world and processes vary often. Mostly, different software applications are used because of optimisation and speed. General software development with bug-tracking system in the behaviour life cycle can be seen in the following figure. This system will develop as long as the software continues. Software –analysis and –architecture using a bug tracking is for any stake-

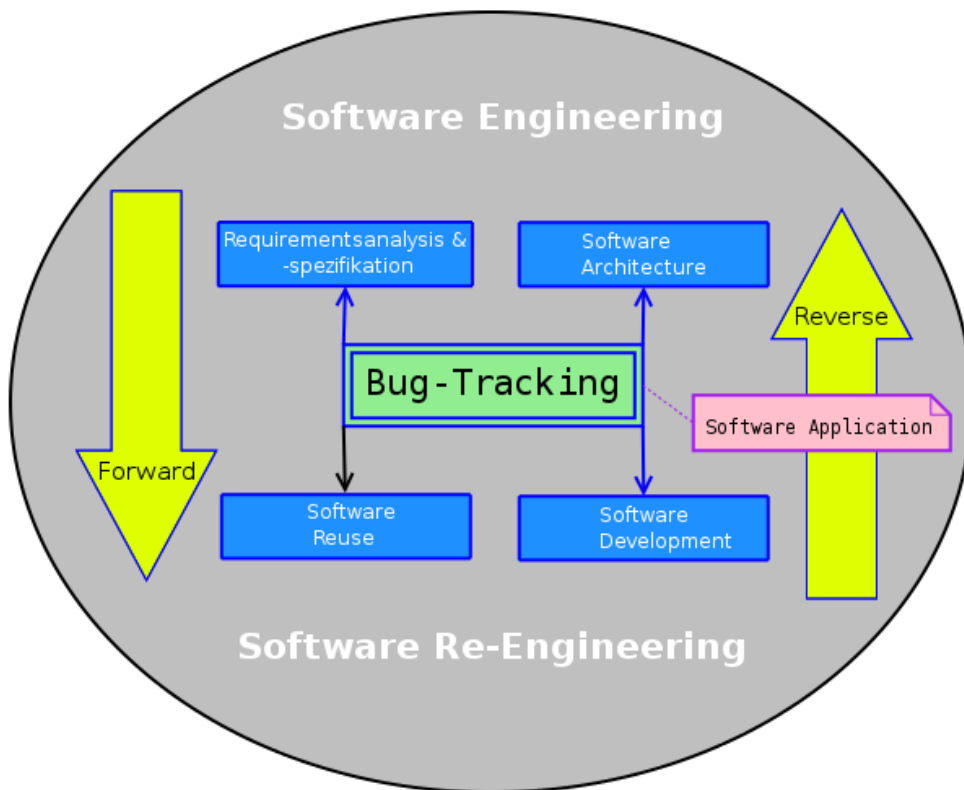


Figure 4.2: Relationship between Software & Bug-Tracking System

holder the core for checking tasks. It is helpful in some areas as for example management implementation, programming and re-documentation where it indicates the attitude between evolution- and maintenance- process and specification. Requirement analysis shows the minimum requirements without bug and can continue with analyzing software project. Software reuse with bug tracking is always applicable. For fast and next smaller projects as the case may be subprojects relevant to examination whether the bug is present. It makes for the user or stakeholders the quick way to identify them. We do not know whether the software incorrect is or not, so the best

method is to apply research with bug-tracking during execution of building the platform e.g. test tracking and the end of the platform application.

Relevant notes for bug tracking are [Ebe05]:

- Before deciding to install a bug-tracking program on your system, try it out on the Internet.
- The correct input of each bug is as important or even more important than bug-tracking system itself.
- Build your project structure and modules well to make it easier for the user and the programmer to identify bugs.
- It is more positive and realistic to think about issues rather than just bugs. If you talk only about bugs, your external viewers will count enhancements as bugs.
- Do not overuse the "send mail to programmer" feature to input bugs and issues.
- Decide whether you want to use your application as a bug-tracking system, a task registry, or both.
- Show your lead users how to introduce effective bug descriptions so that they can debug successfully and explain them well to other users.
- Use the system often to check for new issue reports; otherwise, your customers would not use it. Reply quickly to customer so that they continue to use the system instead of making a phone call to report a bug.
- Use the tool to get information, not to keep tabs on your workers behaviour; otherwise, they would not input true data.

4.2 Bug Tracking Systems

Well-known Features are: [Cod06]

- Search by project, assigned person, priority, status
- Sorting by any of the columns (Bug name, project, priority, assigned Person, status)
- Login authentication
- Administration of users

4.3 Bugzilla

We will describe the structure of the Bugzilla. Bugzilla is implemented software for server and client architecture, particularly for software and testing development. Detailed information can be found under [Moz06].

4.3.1 General description

Bugzilla is the leading open-source/free software bug tracking system, with high-profile installations at Mozilla, Gnome, Red Hat, and Nasa, among others.

The availability of Bugzilla is under software licence as Mozilla project and projects under same integrated the Mozilla Published Licence (MPL), the General Public Licence (GNU) and the Lesser General Public Licence (LGPL). It can be used in different ranges of software applications. It can be used for normal user-to-user (UTU) for errors or in user-to-peer (UTP) software development for reuse, recovery, advancement and evolution of the project. Bugzilla can be used with Web interfaces.

Why actually the name Bugzilla? It gives only the error report messages as integrated structure therefore the name for Bug. It is in the Mozilla – Web Browser integrated, therefore named Zilla.

4.3.2 Features

The features are:

- request system
- comprehensive set of fields
- attachment management
- user wildcard matching
- inter-bug dependencies & time tracking
- a powerful query interface
- generic reporting & -charting
- email notification changes
- power and manual query

- multiple authentication methods
- console interfaces to accomplish
- enterprise group support
- localisation
- full-text search
- patch viewer
- comment reply links
- sanity check

4.3.3 Description

If for Bugzilla an information source is identified with error message, it tries to keep this information. This is stored into a certain data base structure. Communication e.g. connection between bug report and data base runs in the Extensible Markup Language (XML) format. Information and bug report in each case communicate also with XML format. See also the following figure.

It provides administrative information of software and hardware, mailing and carbon copy addresses, keywords for bugging, Uniform Resource Locator (URL) for certain definitions, descriptions and discussions between persons, error to repair in maintenance process.

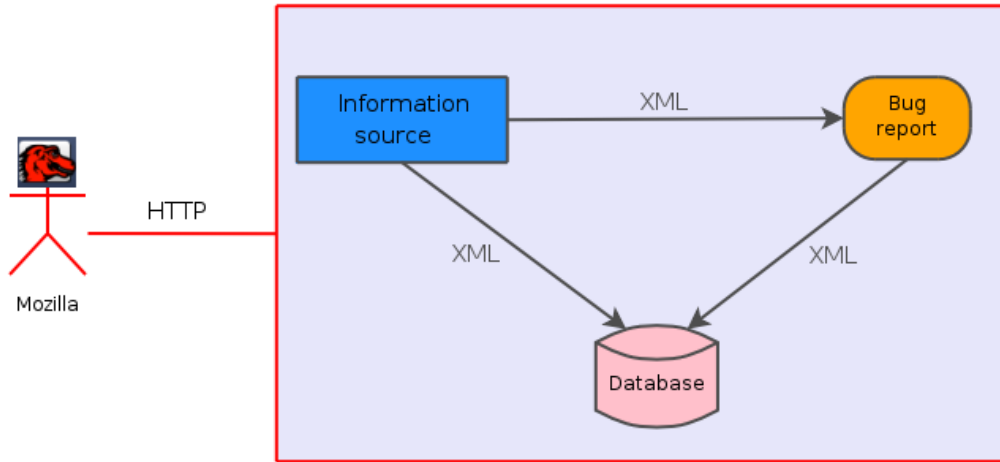


Figure 4.3: Technology architecture

We will represent functionalities of bug report in the next figure. This Figure shows typical Bug report message from Mozilla. It shows us the interpolation and then switching to themes tab crashed minefield. Bug report gives us those message data as Web dialogue. This representation shows us Bugzilla used with a Web Broser in our case Mozilla. The Figure functionalities and attributes are give in a later section with detailed description.

4.3.4 Options

Bugzilla does:

1. Track bugs and code changes
2. Communicate with teammates
3. Submit and review patches
4. Manage quality assurance

Bugzilla can help to get a handle on the software development process. Successful projects often are the result of successful organization and communication. Bugzilla is a powerful tool that can help a team to get organized and communicate effectively.

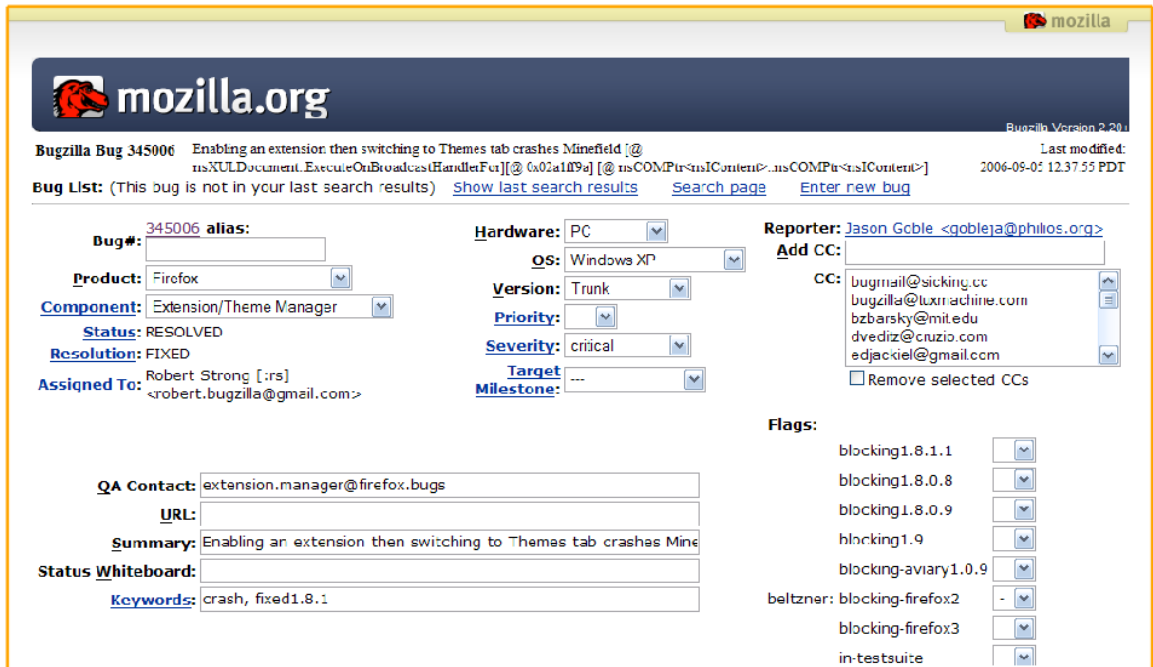


Figure 4.4: Bug-report [Wik06b]

Possible Uses:

- Systems administration
- Deployment management
- Chip design and development problem tracking
- Software and hardware bug tracking
- IT support queues

4.3.5 Design

Bugzilla has potential to become a ticket simple system, task management tool or project management tool but its developers decided to make Bugzilla a software defects tracking system. This figure represents a bug life cycle. It consists of several activity e.g. events, which everyone speaks an independent functionality. They are connected with several dependencies, where

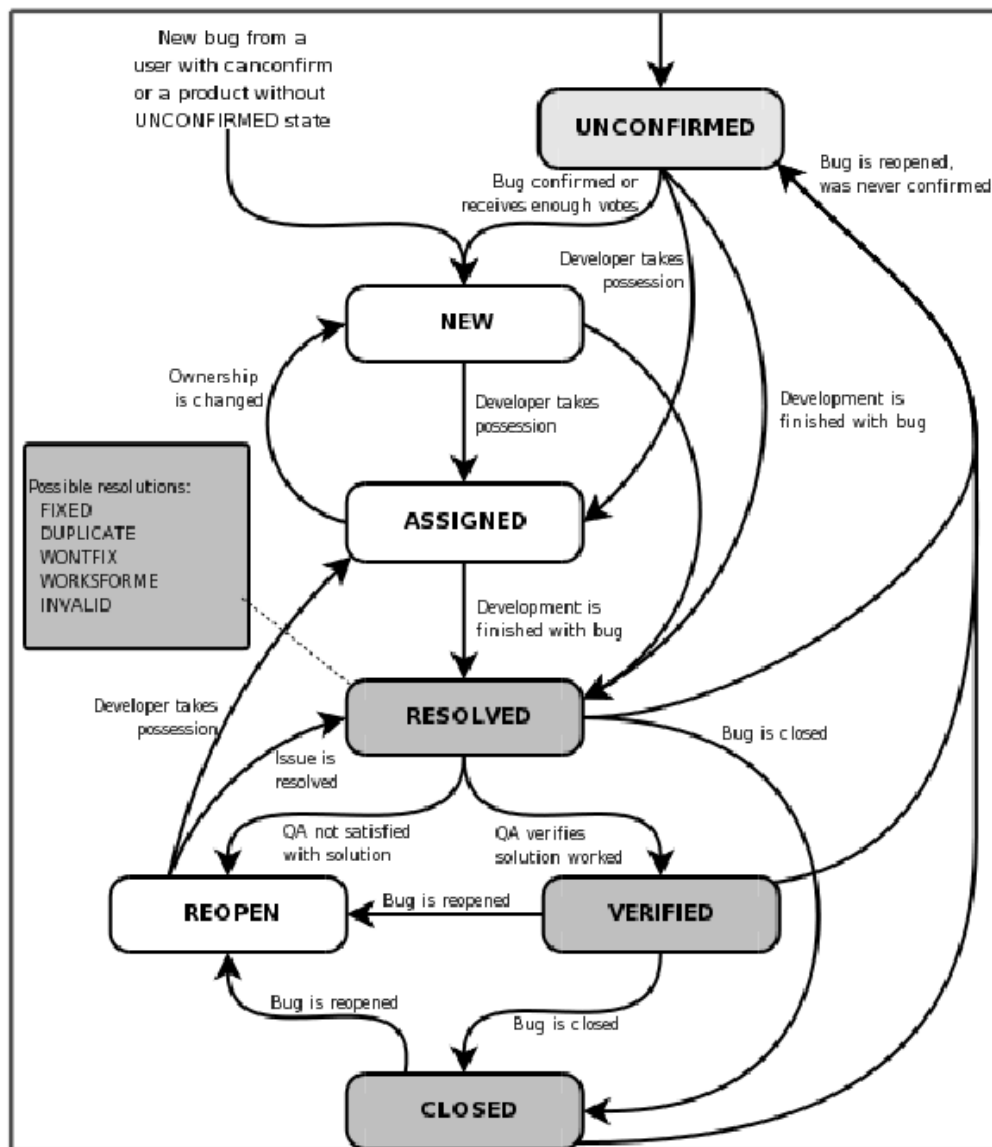


Figure 4.5: Design exposition [Wik06c]

each dependency describes a role attribute.

The states are:

- Unconfirmed
 The bug is not clearly identifiable. It is stored in the data base and respectively marked. If bug revealed to be in a data base as fixed

confirmed and can be accessed at other activities the new, assigned or resolved, independently of its dependency representations.

- New
New bug is found and identified. New bug from a user with canconfirm has be assigned and processed. This activity can accept to assigned state e.g. if the condition is fulfilled or can only be resolved and at the end marked.
- Assigned
This bug is not yet resolved, but is assigned to the proper person [Bug06]. It can go from here at the activity New when dependency Ownership is changed to contend, or to Resolved when Development is finished with bug.
- Resolved
In this activity resolution is taken.
The resolution field indicates what happened to this bug [Bug06].
Possible resolution are:
 - Fixed: Testing for bug into the activities and checked for fixing the bug.
 - Duplicate: Replication of other bugs. If several bugs appear, then replicate bug requires the bug ID of the duplicating bug.
 - Wontfix: The bug will never fixed.
 - Worksforme: All attempts at reproducing this bug were futile, and reading the code produces no clues as to why the described behavior would occur [Bug06].
 - Invalid: The bug has invalid status.
 - Moved: The bug is tracked in another bug data base. It can be removed from another data base.

With the verification Qualification Authority (QA) has the possibility either if QA not satisfied with the solution at the activity Reopen to access or if satisfied with QA then mark as verified.

- Reopen
This activity was resolved, but without success. The resolution is not correct. For example, a Worksforme bug is Reopened when more information shows up and the bug is now reproducible [Bug06]. From here bugs are either marked assigned or resolved [Bug06].

- Verified
QA looked for bug and resolution is taken. Bugs remain in this activity until the product they were reported against actually ships, at which point they become Closed [Bug06].
- Closed
The bug is dead, the activity is closed. The resolution is finally completed. If any bug not correctly worked or the resolution at the end is not completed, then the activity Reopen must be used and the bug processed again.

The requirements are as follows: It needs to be able to run on open source tools. Bugzilla development is supposed to work on commercial data bases, tools and operating systems but it should not happen at the cost of open source tools. Bugzilla's speed and ease characterizing implementation is it's main advantage. It does not call into the data base often so it does not impose heavy load to the database when fetching necessary data and what is also very important it does not generate a Hypertext Markup Language (HTML) code. It avoids Structured Query Language (SQL) calls and data types in queries and tables. Data base specific queries and data types are used when it is possible, thus developers often change existing SQL-Request calls. Browser agnosticism in HTML and form generation, including cleaning up the HTML output of Bugzilla, and following all applicable Web Standards.

4.3.6 Architecture

Bugzilla is Web application, so user and/or stakeholder interact only with HTTP.

4.3.7 Benefits

The advantages are:

- Improved communication
- Increased product quality
- Improved customer satisfaction
- Ensured accountability
- Increased productivity
- Bugzilla can be adapted to multiple situations

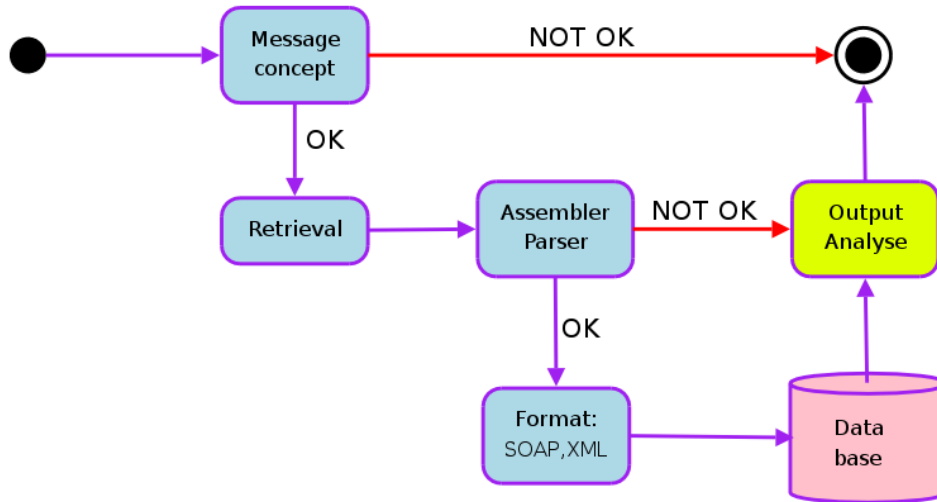


Figure 4.6: Architecture of Bugzilla

4.3.8 Usage

There is a large number of companies, organizations, and projects which use Bugzilla. *Mozilla.org* and the *MediaWiki* projects use Bugzilla to track feature requests. Bugs can be submitted by anybody, and can be assigned to an appropriate developer. A bug can be accompanied by user notes and bug examples.

4.3.9 Data available

Bugzilla in its database possesses separate information, which for users are clearly printed out. This information is stored as attributes the nose status in the data base, where there are by assistance of XML-technology is passed on to the Web-Browser. Here, we present the bug-attributes e.g bug report:

- *id* Clear key and identifier for bug report.
- *product* Describes products for the identified bug-report, e.g the software that contain the bug.
- *component* The description gives the component which with bug is observed and noticed.

- *status* Describe status of bug report. It describes current condition e.g. where condition is now. Different status can follow:

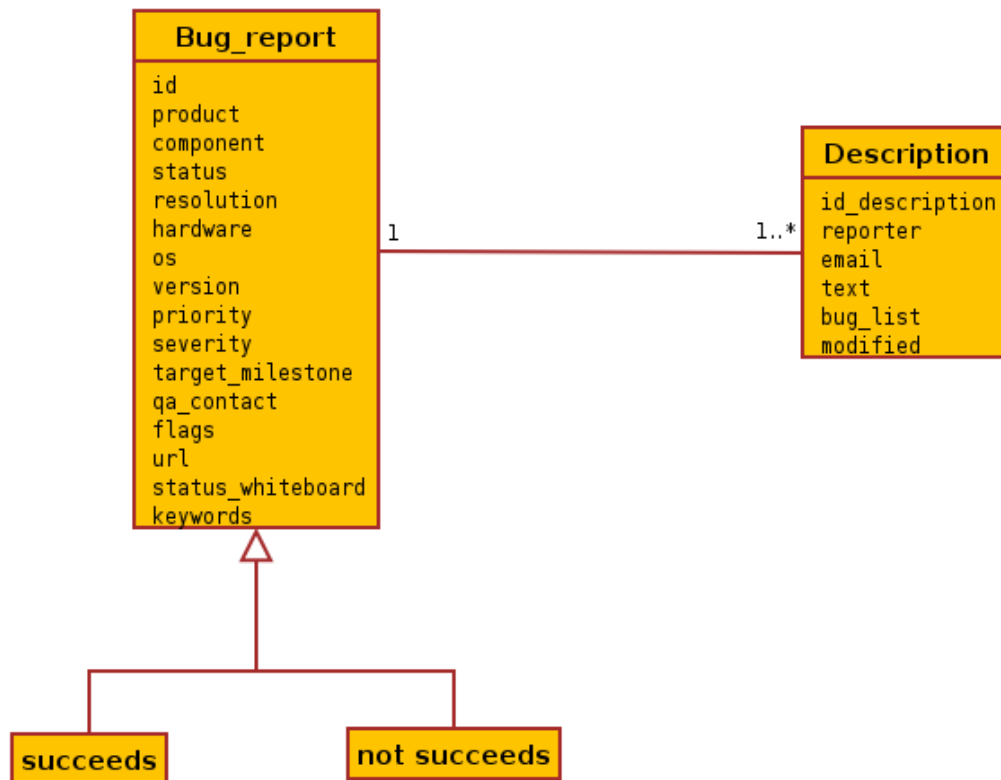


Figure 4.7: Behavior representation with attributes and relations

- resolved
- assigned
- new
- closed
- verifies
- unconfirmed
- needinfo
- opened
- reopened

- *resolution* Shows the actions, which is implemented to the bug. This action can take different status:
 - fixed
 - wontfixed
 - invalid
 - bug
 - notabug
 - obsolete
 - incomplete
 - notgnome
 - notkde
- *hardware* Hardware system describes where errors could lie.
- *os* Option for selection that operating system or architecture systems.
- *version* The version number and description of that product.
- *priority* Priority for error messages. It can taken on:
 - normal
 - high
 - low
 - medium, not high, not low
 - immediate
 - urgent
- *severity* How these errors influences the software. The values are:
 - critical
 - minior
 - major
 - trivial
 - normal
 - trivial
 - enhancement

- *targetmilestone* Possible target versions. The logs file can be passed on to the different targets.
- *qacontact* Contact data by email.
- *flags* Selection criterion for bug flag.
- *url* With the use Bug-tracking discovered a bug per on-line e.g. per Internet.
- *statuswhiteboard* Explain the current state of the bug.
- *keywords* Search profile title of profile description for bug.

The factors which for bug report knowledge are necessary:

- *iddescription* Description the bug with certain inscription.
- *reporter* Name and e-mail address of the bug reporter.
- *e-mail* Description the bug status or confirmation identification by email. It can contain differently elements of the e-mail:
 - From
 - Reply-to
 - To
 - Cc
 - Sender
 - Received
 - Bcc
 - Date
 - In-reply-to
 - Message-Id
- *text* Feedback, remark, suggestions, descriptions for bug etc.
- *buglist* All possible recognized bugs designates with name.
- *modified* Repair, reconstruction and modification bug, so that the Software is fixed.

4.3.10 Studies & Retrieval Tools

We will regard two studies

1. Populating a Release History Database from Version Control and Bug Tracking Systems [FPG03b]
2. Analyzing and Relating Bug Report Data for Feature Tracking [FPG03a]

Populating a Release History Database from Version Control and Bug Tracking Systems

Version control and bug tracking systems enclose a wide number of historical information that gives depth into the evolution of a software projekt. This article describes the population of a Release History Database that connected version and bug tracking report data and shows some dubious examples with respect to software evolution analysis. The basic building blocks are an SQL database and scripts for finding and fittering information from the version control systems and the bug report database. This example shows the Open Source Project Mozilla (OSPM), which uses Concurrent Versions System (CVS)– as version control system and Bugzilla – as bug report database. This article presents how due to Mozilla we can see the collation beetwen results and validate. The weak sides of this system make possibilities to only inadequate of support for a particular analysis of software evolution phase. This problem introduce into a populating a release history database that join version data with bug tracking data and to insert the absent data not to hidden by verion control systems.

The used and tested programs and algorithms are implemented under operating system Unix. The necessary software and programming languages are: CVS, ClearCase, SourceSafe, SQL, Mozilla, Bugzilla, XML, C, C++, Java and Perl.

Analyzing and Relating Bug Report Data for Feature Tracking

This article deals with the closeness of the software features based on modification and problem report data. Engineers are learning from the past from the changes stored in versioning and by tracking systems such as CVS and Bugzilla to determine problem areas and provide for future changes. It is often not very easy because of the number of the amount of recorded data. The modification report (MR), and problem report (PR) fold out of bug data and fix information are the main contribution for the construction of a release history database (RHDB). Transformation of reports are retrieved from the systems such as CVS and PR from bug tracking systems such as Bugzilla.

Problem reports data disagreement matrix are chosen from the RHDB due to Java to validate the SQL queries. Unix is the operating system in this article.

The necessary software and programming languages are: CVS, Bugzilla, Mozilla, GNU, Perl, C, C++, XML, Java and SQL.

4.4 Source Forge Tracker

SourceForge[®] is a software development system. It is integrated with software applications, source code and built for the possibility to amalgamate with other open source software.

Definition 5 *SourceForge is proprietary¹ software. [Wik06d]*

Definition 6 *SourceForge is a collaborative software development management system. [Wik06f]*

SourceForce is deployed by VA Software company. That company is know as Open Source Technology Group (OSTG). The name VA came of two person James Vera and his colleague, graduate student Larry Augustin as last first letters of surnames.

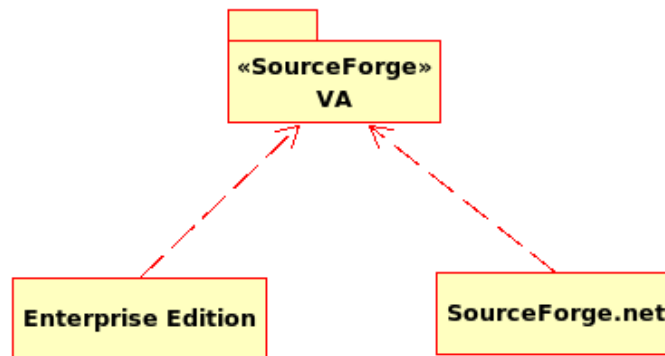


Figure 4.8: Software Tool VA

¹Proprietary software is software that has significant restrictions on using and copying it, usually enforced by a proprietor. [Wik06e]

SourceForge e.g. VA Software from collaborative software development has allowed SourceForge.net (**SF.net**) to develop diverse software. The mission of SF.net is to enrich the Open Source Community (**OSC**) by providing a centralized place for Open Source (**OS**) developers to control and manage OS software development. [Jup06] With general experiences of OSC and of SourceForge.net found out a new product. This product is called SourceForge Enterprise Edition (**SEE**). SEE is an application of VA-Software. This came, from the sense of deficiency and repair functionalities of SF.net. It retains the effectiveness and specter of SF.net. SEE provide enterprise height reliability, performance, reuse, expandability and maintenance.

Definition 7 *SEE is a secure, centralized, enterprise-grade solution for optimizing distributed development. [VA 06b]*

SF.net places a broad multiplicity of services to the projects like hosting. The most well-known services of SF.net are:

- Open Source Software (**OSS**)
- Open Community (**OC**)
- Web Tools for Community (**WTC**)
- File Release System (**FRS**)
- Donation System (**DS**)
- Compile Farm (**CF**)
- CVS Service (**CVSS**)
- Subversion Service (**SS**)
- Communication Tools (**CT**)
- Publicity (**Pub**)
- Project Web Service (**PWB**)

4.4.1 Technology

We will describe point to point of the optimized distributed development with SourceForge. These points are (see also the respective figure):

- Central Repository, assured in such a way core to distributed development
- Global Development Dashboard
- Collaboration
- Process & Controls
- Visibility & Access
- Project Tools
- Interoperability



Figure 4.9: Features SEE [VA 06a]

Central Repository: All functions and characteristics of the optimizing distributed development are complete in the middle point. The features are:

- centralizes information and tools for request and respond, control property assets
- secure
- supporting maintenance and reuse
- back-up data and facilitate administration
- comprehensive architecture and integration with existing systems

Global Development Dashboard: provides real-time project status and statistics for visibility and insight into projects. It provides detailed reporting on application, project or tool level.

Collaboration: accessible per Web browser. It provides view into disparate project and development. Collaborative Development Environment include: [VA 06c]

1. Collaborative Development Environment
2. Document Manager
3. Discussion Forums, Mailing Lists and News

Process & Controls: impact on secure workflow control and configurable process. It provides:

1. Secure access
2. SourceForge collaborative Development Process
3. Automated monitoring, assignment and notifications
4. Traceability/compliance

Visibility & Access: represents the visibility and the application. It provides:

- reusable assets
- documents
- source code
- issue & task details
- discussions & emails

- audit logs

Project Tools: provides EtoU, web appliance and used for task tracking, issues and software development lifecycle. Project tools are:

1. Tracker
2. Task manager
3. File release system

Interoperability: Supports integration and interoperability with SourceForge enterprise systems. This Interoperability integrates in two ways: [VA 06d]

- interoperability with used project planning and management tools, office productivity applications, software configuration management (SCM) systems, plus integrated development environments (IDEs)
- integration-ready architecture and open API that make it uniquely well-suited for integration, extensibility and use within complex enterprise environments

4.4.2 Architecture

SourceForge gives an integrationable architecture, this was explicitly sketched for software technology integration, developable, surely and simply to use.

4.4.3 Tracker

Definition 8 *Tracker is used in the software development and serves for the collection and documentation of program errors.*

With tracker it is possible to write feature reports. It can be used for economic and technical purposes. Options for tracker are:

1. *Assignee* : tracker item is assigned and choosen. Assignee can be:
 - *any*
 - *unassigned*
2. *Status* : represents the current situation of a tracker item. There are five possibilities to choose from:
 - *any* – Arbitrary status.

- *open* – Status is opened to read.
 - *closed* – Status message is closed.
 - *deleted* – Message or status report is deleted.
 - *pending* – Used, when the person who sent the message waiting for the reply. After some time the status changes automatically to *open*. But if the person does not respond to the server for fortnight, the status becomes *deleted*.
3. *Category* : to choose the different category of the tracker. Different possibilities are:
- *any*
 - *3rdPartyPlugin*
 - *general*
 - *interface*
 - *protocol*
4. *Group* : is selection of the Group with the tracker items. There is any possibility to select group:
- *any*
 - *v1.0*
 - *v2.0*
 - *v2.5*
5. *Sortby* : is choose how do you want to sort the results.

It is also possible and accomplishable that independently any user can define and use personal tracker item. There are clear pre-defined methods within SourfeForge where some tracker item is be placed.

The most open source programs hosted on Sourceforge.net use tracker bug to describe bugs. Tracker bug describes functionalities of software that are incorrect. It possesses the attributes and methods. Attributes give the description of tracker bug:

- Request ID
- Summary
- Open Date

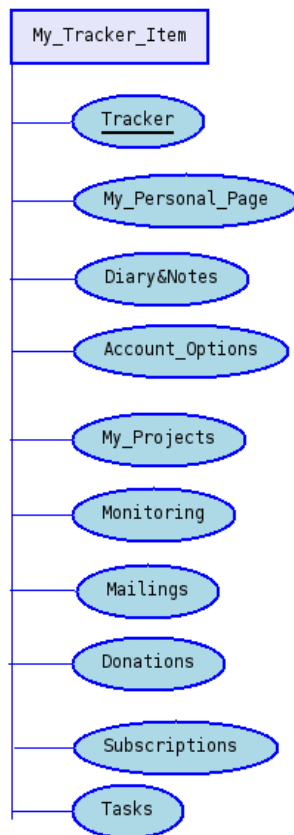


Figure 4.10: Tracker Item demonstration

- Priority
- Assigned to
- Submitted By

We described methods with tracker. See options of tracker.

4.5 GNATS

GNATS is the GNU Bug Tracking System. It is a software system with either graphical or command line shell interface.

Definition 9 *GNATS is set of tools for tracking bug reports. [Fre06b]*

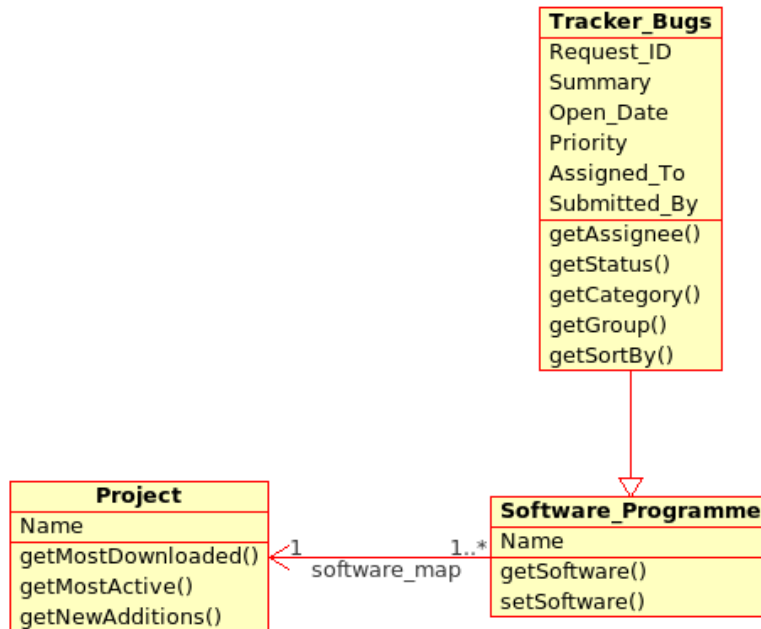


Figure 4.11: SourceForce Tracker

GNATS is a bug-tracking program assembly for using and finding bugs. It keeps all information about reported errors in the database and provides tools for searching, editing, using, repeating and managing this database.

In the view of the GNU project GNATS is a Report Management System (RMS). GNATS stores all information about the problem reports at a central site, and enables users to access this site by various means, including e-mail, www, and a network daemon. [Fre00a]

When any problem comes up during using the software, users enter problem either via emails to the maintainers or make contact with a GNATS server.

4.5.1 Architecture

The following figure describes communication between server and client. Server starts with a choice server like inetd or xinetd. It has a built-in access control mechanism based on IP² addresses and username/password [Fre00c]. If the server is started with xinetd³, then create a file `/etc/xinetd.d/support`

²Internet Protocol

³eXtended INTErnet Daemon

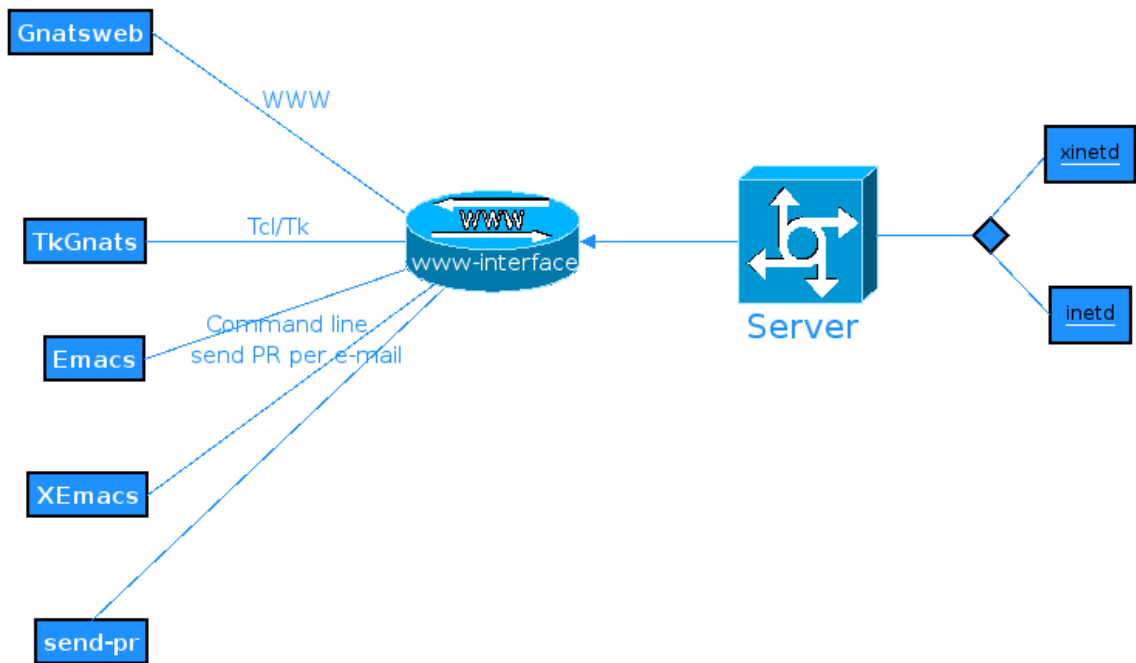


Figure 4.12: GNATS Architecture

(see also the respective figure).

Server configuration under inetd⁴ reads as follows:

```

service support
{
    disable      = no
    socket_type  = stream
    protocol    = tcp
    wait        = no
    user        = gnats
    server      = /usr/local/libexec/gnats/gnatsd
               server_args = gnatsd
}

```

Figure 4.13: Xinted Configuration [Fre00d]

⁴Internet Daemon

1. file to call under `/etc/inetd.conf` and than editing


```
#port      userid program
support stream tcp nowait gnats /usr/local/libexec/gnats/gnatsd gnatsd
```
2. call file under `/etc/services`

```
support      1529/tcp      # GNATS
```

The discussion between server and client is completely by `www-interface`. `www-Application` occurs under GNATS default port 1529. If you want to run GNATS under another port, then change default port to another port. Most clients also accept an option or configuration variable to change the port. Do not forget to tell `inetd` or `xinetd` to start `gnatsd` on the other port. [Fre00e]

Well-known client distributions from GNATS are:

- Gnatsweb
- TkGnats
- Emacs
- XEmacs
- `send-pr`

Gnatsweb has a WWW interface, which gives requests to the server.

TkGnats has a Tcl/Tk based interface. It is capable of contacting several GNAT servers or several problem report databases managed by the same GNATS server. [Fre00b]

Emacs, XEmacs and `send-pr` are GNATS mode and have a traditional command line interface in which any problem report can be sent by e-mail.

Emacs and XEmacs are text editors. XEmacs was founded by GNU Emacs, but has a different philosophy from Emacs development. It is more open to experimentation, and offer as first the new features, such as inline images, variable fonts and terminal coloring.

`send-pr` is send problem report (SPR) to a central support server site. The `send-pr` in NetBSD system is the great user-interface and design for bug notifications to a GNATS database. After completing the formular SPR sends the report to the GNATS server per e-mail. When it arrives the syntax is first checked, and then becomes a new problem report (PR) number and catalogue to the GNATS database. The sender receives a statement with the PR number.

4.5.2 GNATSweb

GNATSweb (GNATwww) is a web front-end to gnats, the GNU Problem Report Management System. It is a CGI⁵ script which runs on any web server. GNATwww is similar to wwwgnats. It uses gnatsd, which means that the web server and gnats server do not have to be on the same machine or have access to the network. GNATwww does not use nquery-pr, it contacts gnatsd directly.

The screenshot shows the 'Query Problem Reports' interface of GNATSweb. At the top, there is a navigation bar with 'support' on the left and 'gnatsweb' on the right. Between them are links for 'MAIN PAGE', 'CREATE QUERY', 'ADV. QUERY', 'LOGIN AGAIN', and 'HELP'. The main heading is 'Query Problem Reports'. Below this, there are two buttons: 'submit stored query' and 'delete stored query', each with a dropdown menu currently set to 'all open critical'. A 'submit query' button is also present. The form fields include: 'Category' (dropdown: 'cluster-btree - B-tree and interrupt management'), 'Severity' (dropdown: 'serious'), 'Priority' (dropdown: 'all'), 'Responsible' (dropdown: 'all'), 'State' (dropdown: 'all'), 'Class' (dropdown: 'swtbug'), 'Synopsis Search' (text input), 'Multi-line Text Search' (text input), 'Column Display' (dropdown menu showing 'category', 'confidential', 'state', 'class', 'severity'), and checkboxes for 'Ignore Closed' (checked) and 'Originated by You' (unchecked). A 'Display Current Date' checkbox is also checked. A 'submit query' button is at the bottom of the form.

Figure 4.14: Query Problem Reports [Ope06a]

4.5.3 TkGnats

TkGnats is a graphical frontend for the free bug tracking system GNATS based on the Tcl/Tk for the GNATS bug management system. It runs on UNIX and Windows 95/98/NT in difference from GNATS which runs only on UNIX platforms.

⁵Common Gateway Interface

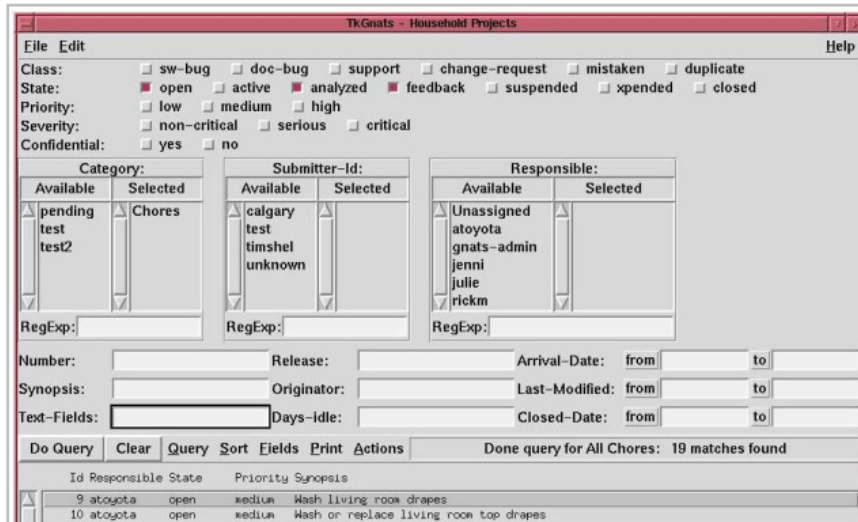


Figure 4.15: TkGnats–Household Projects [Ope06b]

4.6 PHP Helpdesk

PHP Helpdesk (PHPHD) is an organisation. The licence is GNU GPL. It is a customer support system written in PHP and using MySQL database backend. It is a project to create a PHP–based bug–tracking system.

PHPHD was developed because of my need to keep a list of jobs that were pending. I have tried all sorts of other helpdesk related tools and there was nothing that I have found that would handle the problems that I had. It requires PHP, MySQL and a web server. [PHP06a]

PHPHD is an online helpdesk program for use in organizations with following features:

- Bug tracking
- Insert user/company
- Remove user/company
- Revision user/company

- Authorize information
- Identification of job in use
- Closing the job
- Produce request on response time and to identify bugs with each computer.



Figure 4.16: PHP Helpdesk [PHP06b]

This figure shows us how can entered some Project by tracking bug.

4.7 PHPBugTracker

PHPBugTracker (PHPBT) is a web-based bug tracking system. It is similar to tracking system BugZilla. See also chapter 2.1.

This software helps any development to manage testing and debugging.

If there is more than one development project e.g. web application for which you can identify a bug, you will be questioned to choose one.

This figure describes a noticed bug. The functionalities are:



Figure 4.17: phpBugTracker: Enter Bug Project [php06d]

- home
- add a new bug
- query bugs
- view reports
- create a new account
- read documentation
- admin tools

The methods are:

- status

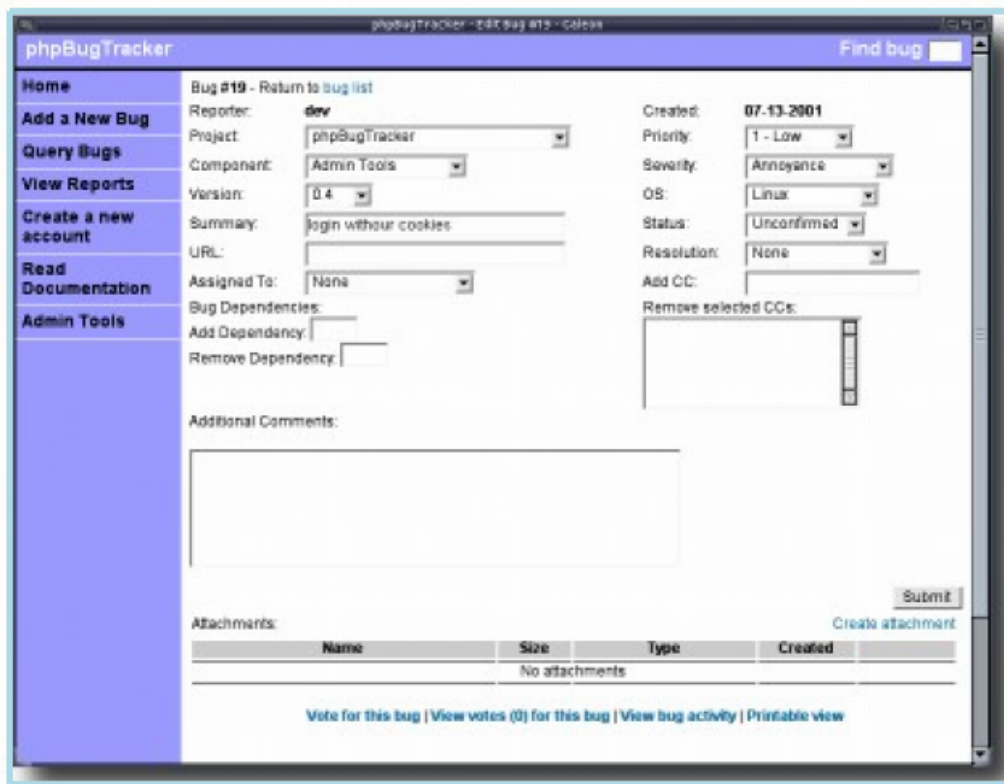


Figure 4.18: PHPBugTracker Bug [php06c]

- resolution
- operative system
- priority
- severity

Adding comments to a bug allows you to provide more detail about what caused the bug or what the expected behavior was, but also allow the developer and the user to communicate about a bug while keeping a history of those notes. Comments that are added to a bug will be emailed as described previously [php06c].

PHPBT sends email to a certain user, who identifies itself with bug. If a bug has been submitted and assigned to a developer, and then the developer

adds comments to the bug, then those comments will be emailed to the reporter but not to the developer. [php06c] Additional people can be added to receive these change emails by adding them to the CC⁶ list. To remove people from the CC list, simply select the people to be removed and submit the form [php06c]. Create Attachment link adds bugs as attachment files. Attachments can be viewed from the attachment list.

PHPBT runs under a simple architecture conception. See also the respective figure. Web-Server (Apache, Tomcat, JBoss etc.) need to be configured to run PHP programme and scripts. PHP⁷ is a programming language with a similar syntax as Perl, which is used mostly for the dynamic environment by web pages or applications of Web. Database runs under MySQL, PostgreSQL and Oracle. It can run also under other databases only the installation is not working so simply.

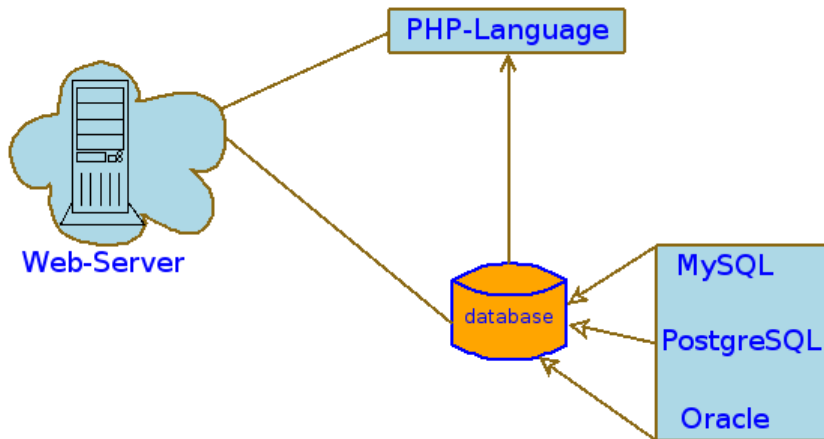


Figure 4.19: PHPBugTracker Architecture

4.8 Double Choco Latte

Definition 10 *Double Choco Latte (DCL)* is a GNU Enterprise package that provides basic project management capabilities, time tracking on tasks, call

⁶Carbon Copy

⁷Personal Home Page Tools

tracking, email notifications, online documents, statistical reports, report engine, and more features either working or being developed/planned. [Fre06a]

DCL is a system for tracking bugs, changes, requests for software, reuse of project. It can communicate also with several clients, e.g. multiple Clients(request) and server(respond) communication.

DCL is a project to create a solution for managing some IT departments including software development and call center activity. It has a web interface and will also have a stand-alone Java client. [Ope06c]

DCL goes into more depth than PHPHD tracking tool, therefore concerns internal features more deeply:

- the strong investigation for bug
- very good discovery for bug
- runs under *Java*TM technologie

The user can search for bug (see respective figure). User opens graphic dialogue search menu and starts searching. There are options for searching the bug:

- New work order
- New project
- New ticket
- Main wiki
- Printer Friendly

User has different methods to select.

- summary
- personnel
- type
- product
- project

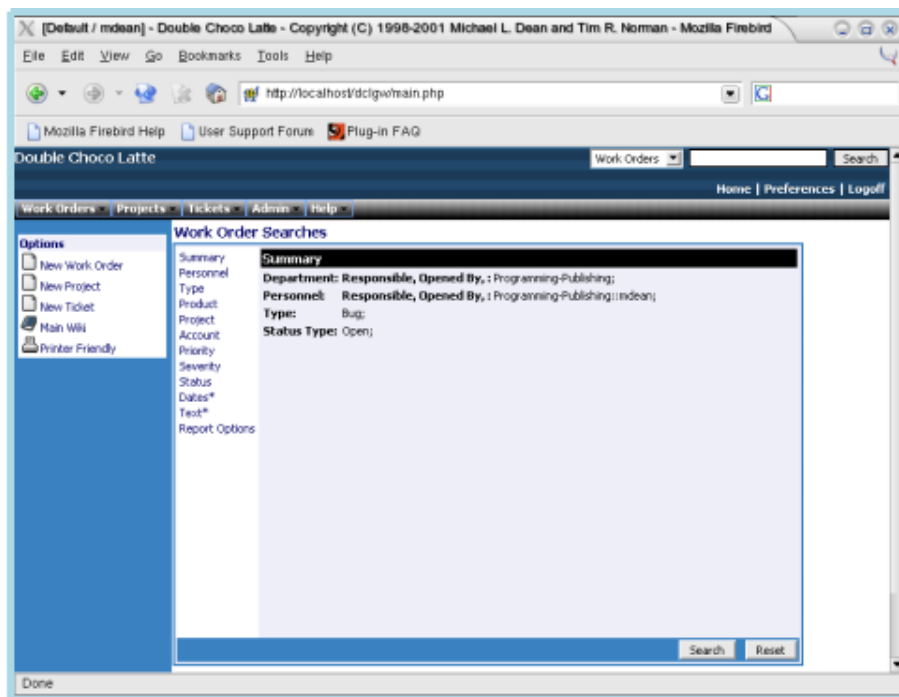


Figure 4.20: Search menu [Dou06b]

- account
- priority
- severity
- status
- dates
- text

This figure shows the results. Found bug gives detailed description as:

- WON.⁸
- Sequence
- Responsible
- Product

⁸Work Order Number

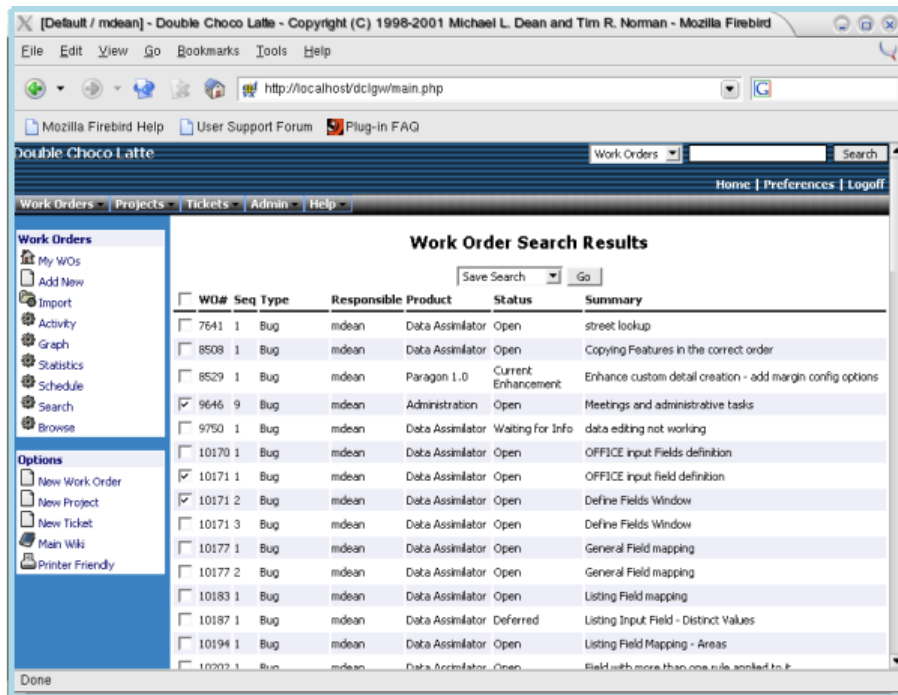


Figure 4.21: Search result [Dou06a]

- Status
- Summary

There exist different options to regulate:

- My WOs⁹
- Add new
- Import
- Activity
- Graph
- Statistics
- Schedule
- Search

⁹Work orders

- Browse

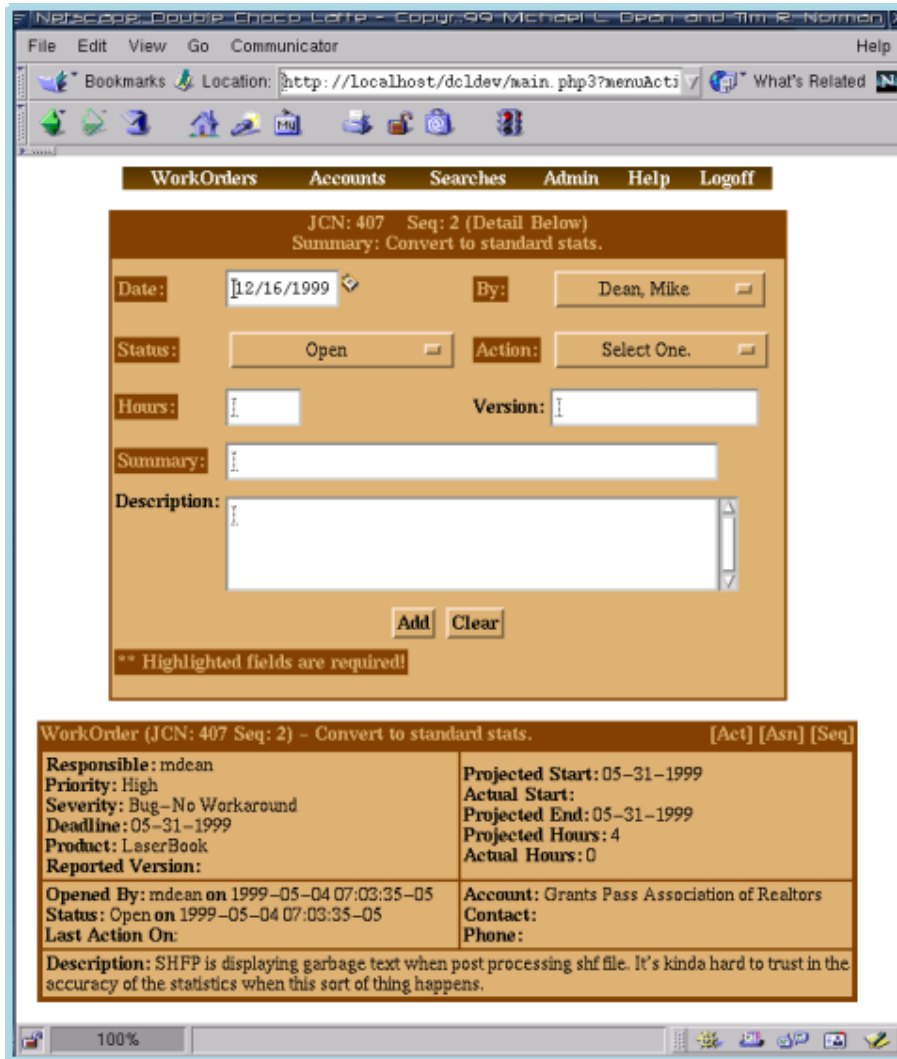


Figure 4.22: Action [Dou06c]

The User tries to use an action. There is selection different action to define and to regard. In the Work Order there is possibility certain actions to select. The selected action depends on user. This figure represents action in the attacker bug identified and found. The dialogue action shows us the possibilities for using:

- Date

- By
- Status
- Action
- Hours
- Version
- Summary
- Description

Functionalities in this figure are:

- Add
- Clear

The action dialogue gives a report of whether the action implemented was effective or not.

1. Responsible
2. Priority
3. Severity
4. Deadline
5. Product
6. Reported Version
7. Projected Start
8. Actual Start
9. Projected End
10. Actual Hours
11. Opened by
12. Status
13. Last Action on

14. Account
15. Contact
16. Phone
17. Description

Appendix A

Data Sources

This appendix shows several matrices where each of the potentially exploited categories of community-related information is subdivided into each of the possible types of repositories.

Six categories have been defined and for each one, several types of repositories have been identified.

In order to facilitate the comprehension of next tables, each type of repository has been matched with a color. Three colors have been used, red, yellow and green. If there are no tools available to extract information from a given repository, the color is red. Just the opposite represents the green color and finally, the yellow one indicates that a tool is being developed to extract information from it.

Most of the repositories show a red color, what means that there is no tool available to extract data. However, the main forges, like SourceForge, Berlios or Savannah just provides support for one or two of them. For instance, focusing on the source code management system, CVS or Subversion are the most used in FLOSS projects. Also, some other distributed source code management systems, like Git, Mercurial or Bazaar, what means that data can be extracted from most of the aforementioned repository.

Also, around half of the repositories tools is licensed under the GPL, BSD or any other floss license. Thus, what is presented here is a list of repositories found for each category, but also the availability of tools to extract information, at least from the FLOSS world and with FLOSS tools.

Summarizing next figures: figure A.1 shows a list of the several types of repositories found for source code management systems¹. Figure A.2 and figure A.3 show a list of issue tracker systems and bug tracker systems².

¹Based on http://en.wikipedia.org/wiki/Comparison_of_revision_control_software

²Based on http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_

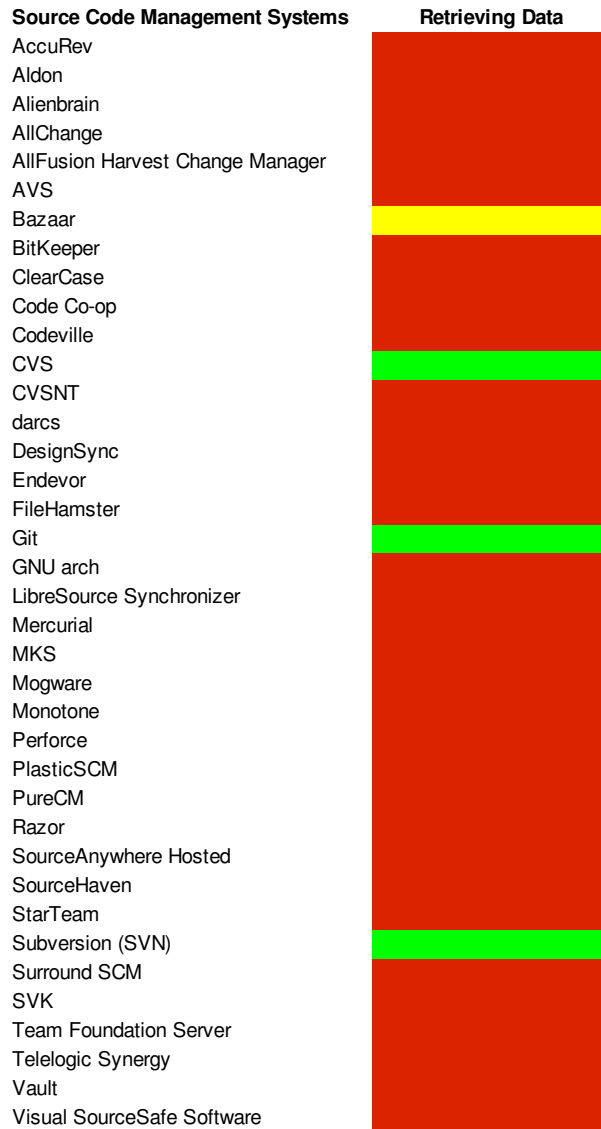


Figure A.1: Types of source code management systems and the availability to extract data from them

Issue Tracking Systems	Retrieving Data	Issue Tracking Systems	Retrieving Data
Atadesk		TagTicket	
A.InformUP		Target Helpdesk Software	
ConSol CM		Team Foundation Server	
DevTrack		Mojo Helpdesk	
doTask!		OTRS	
Eventum		Polarion ALM Enterprise	
ExtraView		Remedy Action Request System	
fixx		Request Tracker	
FogBugz		Roundup	
GLPI		ServiceCenter	
h2desk		SharpForge	
HelpSpot		Simpleticket	
IBM Rational ClearQuest		Spiceworks Desktop	
Instant Business Network		TagTicket	
IssueNet		tBits	
IssueTrackerProduct		Teamwork (software)	
Issuetrak		Tele-Support HelpDesk (software)	
Issue Tracking Anywhere		ThinMind.com	
IssueView		Tracker	
Liberum Help Desk		Unicenter Service Desk (USD)	
LibreSource		VisionProject	
k Systems Management Appliances		Web Help Desk	
Open Project Manager		Wrike	
QueWeb		XMsuite	
SupportSuite		Zendesk	

Figure A.2: Types of issue tracker systems and the availability to extract data from them

Bug Tracking Systems	Retrieving Data	Bug Trackin Systems	Retrieving Data
16bugs		IssueNet	
A.InformUP		IssueView	
AceProject		JIRA	
AVS woodpecker issue tracker		Mantis	
Arctic		Mercury Quality Center	
BugAware		OnTime	
BUGS - the Bug Genie		pragma::tims	
BugSentry		phpBugTracker	
BugTracker.NET		Projistics	
BugWiki		NetResults Tracker	
BugZap		Quality Assurance Studio	
Bugzero		Redmine (home page)	
Bugzilla		Retrospectiva	
Clarity		Scarab	
Collaboa		SourceForge Tracker	
CompassTMS		TeamTrack	
CVSTrac		StarTeam	
Debbugs		TestDirector	
Defect Manager		TestTrack Pro	
Defect Tracker		Teton Suite	
DisTract		ThinMind.com	
DITrack		Trac	
EmForge		Track+	
eTraxis		TrackRecord	
Eventum		TrackStudio Enterprise	
FIT-BugTrack		Unfuddle	
fixx		VisionProject	
Flyspray		Volo Fixer	
FogBugz		WorkRoll	
Fortress		Wrike	
Gemini		XMsuite	
GNATS		yKAP	
IBN Help Desk		XStudio	

Figure A.3: Types of bug tracking systems and the availability to extract data from them

Figure A.4 shows the types of mailing lists and the availability to extract data from them. Most of the mailing lists or similar tools (like forums) are based on HTML format. So far, just mbox format is analysable.

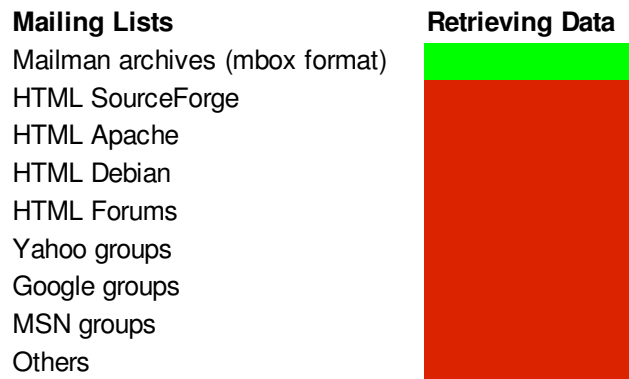


Figure A.4: Types of mailing lists and the availability to extract data from them

Regarding the source code, figure A.5 shows a list of the analysable source code. Extra source code files are also analysable, however the tools have not been included in the FLOSSMetrics platform.

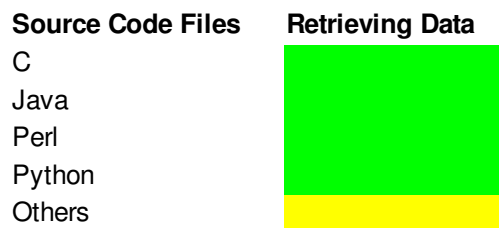


Figure A.5: Types of programming languages to be analysed

Finally, wikis are an important tool to document projects, however in this side, tools are not developed and most of the analysis would include semantic analysis, what is more complicated than the automatic analysis addressed before.

systems

Wiki	Retrieving Data
MediaWiki	
MoinMoin	
TikiWiki	
Twiki	
Others	

Figure A.6: Types of wikis and the availability to extract data from them

Appendix B

Tools and Metrics

B.1 Tools

Tools to be integrated in the FLOSSMetrics platform are next:

- **CVSAnalY and CVSAnalY2** - They extract information out of CVS, Subversion or Git repositories logs and transforms it in a database SQL format.
- **Mailing List Stats** - tool for mapping mbox files of any mailing list to a database.
- **Bicho** - It stores information from a given bug tracking system to a database format. So far, it only works with SourceForge BTS.
- **SLOCCount** - This is a suite of programs for counting physical source lines of code (SLOC) in large software systems. It can count physical SLOC for a wide number of languages and also it can take a large set of files. It provides some other analysis tools. Many projects are developed in more than one programming language. Using SLOCCount, it is possible to quickly assess the percentage of source lines of code for each used language. SLOCCount also computes the estimated effort based on a COCOMO estimate. It is possible to customize COCOMO parameters on the command line.
- **PyMetrics** - It provides metrics for Python programming language. For instance McCabe's Cyclomatic Complexity metric, LoC, Comments, etc. Users can also define their own metrics using data from PyMetrics. PyMetrics outputs SQL command files and CSV output.

- **CCCC** - C and C++ Code Counter) analyzes and reports measurements on source code written in C, C++, and Java.

Metrics supported include lines of code, McCabe's complexity and metrics proposed by Chidamber&Kemerer and Henry&Kafura. CCCC computes the following metrics: Number of Modules, Lines of Code, Lines of Comments, an approximation of McCabe's Cyclomatic Complexity, Information Flow (coupling between modules), Weighted Method per Class, Depth of Inheritance, Number of Children, Coupling between Objects, Fan In and Fan out.

- **PerlMetrics** - This tool provides metrics as counts lines, packages, subs and complexity of Perl files.

B.2 Metrics

The next set of metrics are categorized by data source.

- **Source Code Metrics.**

Language, number of source lines of code, number of lines of code, number of lines of comments, number of blank lines, number of functions, complexity metrics, like mccabe_max, mccabe_min, mccabe_sum, mccabe_mean, mccabe_median and halstead metrics, like halstead length, halstead_vol, halstead_level and halstead_md.

- **Source Code Management Repository.** Per each of the next metrics, the mean, median, maximum, minimum, quartiles and standard deviation will be calculated. Metrics can also be obtained dividing by periods like month or year.

Size of files(CVS repositories), total time spent in the project by a developer, number of modifications (commits) in a file, time worked on a file (difference between the last and first commit in that file), number of active developers, number of commits, number of LOC (CVS repositories)(added and deleted), number of authors working on the same file, number of authors working on the same file, number of revisions per committer and number of files modified per author.

- **Mailing Lists.** Per each of the next metrics, the mean, median, maximum, minimum, quartiles and standard deviation will be calculated. Also the data will be divided into periods of months or years.

Number of posters, number of posters per mailing lists, number of posts per author, number of posts per author and mailing list, length of the thread, number of replies, number of replies per distinct author and number of replies per mailing list.

- **Bug Tracking System.** Per each of the next metrics, the mean, median, maximum, minimum, quartiles and standard deviation will be calculated. Also the metrics will be divided by periods, as month or year.

Number of bugs per Status, number of bugs per submitter, number of bugs per developer working on (assigned to), resolution time, number of comments, number of changes (a change is carried out when one of the fields is modified), number of people commenting in the same bug, number of bugs per priority, backlog management index and percentile of delinquent fixes

Bibliography

- [BR05] A. Bonaccorsi and C. Rossi. Collection of activity data for sourceforge projects. Technical Report: TR-2005-15, University of Notre Dame, 2005.
- [Bug06] Bugtracking. A bug's life cycle. <http://dev.processing.org/bugs/page.cgi?id=fields.html>, Mit Dez 6 15:48:47 CET 2006.
- [CO006] *Beyond Low-Hanging Fruit: Seeking the Next Generation in FLOSS Data Mining.*, 2006.
- [Cod06] CodeCharge Studio. Conception Bug Tracking System. http://www.gotocode.com/apps.asp?app_id=1, Mit Dez 6 12:09:51 CET 2006.
- [Deu04] Rebecca L. Deuel. Automated bug tracking: The Promise and the Pitfalls. *IEEE Software*, 21(1):100–103, 2004.
- [Dou06a] Double Choco Latte. Screenshot Double Choco Latte Searchresult. <http://dcl.sourceforge.net/ss/searchresult.png>, Mon Dez 18 13:45:01 CET 2006.
- [Dou06b] Double Choco Latte. Screenshot Double Choco Latte Searchmenu. <http://dcl.sourceforge.net/ss/searchmenu.png>, Mon Dez 18 13:45:07 CET 2006.
- [Dou06c] Double Choco Latte. Screenshot Double Choco Latte Action. <http://dcl.sourceforge.net/ss/action.png>, Mon Dez 18 14:09:46 CET 2006.
- [Ebe05] Christof Ebert. Bugzilla, ITracker and Other Bug Trackers. *IEEE Software*, 22(2):11–13, 2005.

- [FPG03a] Michael Fischer, Martin Pinzger, and Harald Gall. Analyzing and Relating Bug Report Data for Bug Feature Tracking. In *Proceedings of the International Conference on Software Maintenance*, pages 90–99, 2003.
- [FPG03b] Michael Fischer, Martin Pinzger, and Harald Gall. Populating a Release History Database from Version Control and Bug Tracking Systems. In *Proceedings of the International Conference on Software Maintenance*, 2003.
- [Fre00a] Free Software Foundation. What is GNATS? <http://www.gnu.org/software/gnats/doc/gnats-faq-4.1.999/gnats-faq.html#What-is-GNATS>, Mit Dez 27 14:26:15 CET 200.
- [Fre00b] Free Software Foundation. How it Works - User's View. http://www.gnu.org/software/gnats/doc/gnats-faq-4.1.999/html_node/User-view.html#User-view, Mit Dez 27 16:10:18 CET 200.
- [Fre00c] Free Software Foundation. How it worksadministrator's view. http://www.gnu.org/software/gnats/doc/gnats-faq-4.1.999/html_node/Administrator-View.html#Administrator-View, Mit Dez 27 16:54:49 CET 200.
- [Fre00d] Free Software Foundation. How to configure xinetd to start gnatsd. http://www.gnu.org/software/gnats/doc/gnats-faq-4.1.999/html_node/xinetd-configuration.html#xinetd-configuration, Mit Dez 27 17:05:18 CET 200.
- [Fre00e] Free Software Foundation. Gnatsd port number. http://www.gnu.org/software/gnats/doc/gnats-faq-4.1.999/html_node/Port-Number.html#Port-Number, Mit Dez 27 18:00:56 CET 200.
- [Fre06a] Free Software Foundation. Double Choco Latte. <http://dcl.sourceforge.net/>, Fre Dez 15 14:01:47 CET 2006.
- [Fre06b] Free Software Foundation. GNATS - Summary. <http://savannah.gnu.org/projects/gnats>, Mit Dez 27 13:05:21 CET 2006.

- [GBPdlHQ⁺01] Jesus M. Gonzalez-Barahona, Miguel A. Ortuno Perez, Pedro de las Heras Quiros, Jose Centeno Gonzalez, and Vicente Matellan Olivera. Counting potatoes: the size of Debian 2.2. *Upgrade Magazine*, II(6):60–66, December 2001.
- [Ger03] Daniel German. The gnome project: A case study of open source, global software development. *Software Process: Improvement and Practice*, 8(4):201–215, 2003.
- [Ger04] Daniel M. German. Mining CVS repositories, the softChange experience. In *Proceedings of the International Workshop on Mining Software Repositories*, Edinburgh, UK, 2004.
- [Gho04] A.R. Ghosh. Clustering and dependencies in free/open source software development: Methodology and tools. *First Monday*, 8(4), 2004.
- [HC04] J. Howison and K. Crowston. The perils and pitfalls of mining sourceforge. In *26th International Conference on Software Engineering, Edinburgh, Scotland, 2004*.
- [HK05] M. Hahsler and S. Koch. Discussion of a large-scale open source data collection methodology. In *Proceedings of the 38th Hawaii International Conference on System Sciences (IEEE, HICSS '05-Track 7), Jan 03-06, Big Island, Hawaii, page 197b.*, 2005.
- [HM05] Koch S. Hahsler M. Discussion of a large-scale open source data collection methodology. In *Proceedings of the 38th Hawaii International Conference on System Sciences (IEEE, HICSS '05-Track 7), Jan 03-06, Big Island, Hawaii, page 197b.*, 2005.
- [HS03] Kieran Healy and Alan Schussman. The ecology of open-source software development. Technical report, University of Arizona, USA, January 2003.
<http://opensource.mit.edu/papers/healyschussman.pdf>.
- [IEE06] IEEE Organization. Bugzilla, ITracker, and other bug trackers. <http://ieeexplore.ieee.org/search/freesrabstract.jsp?arnumber=1407819&isnumber=30525&punumber=52&k2dockey=>

1407819@ieeejrns&query=%28bug+tracking%29+%3Cin%3E+metadata&pos=0, Mit Dez 6 08:39:44 CET 2006.

- [JMC06] Howison J., Conklin M., and Crowston. Flossmole: A collaborative repository for floss research data and analyses. *International Journal of Information Technology and Web Engineering*, 1:17–26, 2006.
- [Jup06] Jupitermedia Corporation. All About Open Source. http://www.webopedia.com/DidYouKnow/Computer_Science/2005/open_source.asp, Mit Dez 13 10:07:10 CET 2006.
- [KK04] Joon Koh and Young-Gul Kim. Knowledge sharing in virtual communities: an e-business perspective. *Expert Syst. Appl.*, 26(2):155–166, 2004.
- [KSL03] G. Krogh, S. Spaeth, and K. Lakhani. Community, joining, and specialisation in open source software innovation: a case study. *Research Policy*, 32:1217–1241, 2003.
- [LvH03] K. Lakhani and E. von Hippel. How open source software works: "free" user-to-user assistance. *Research Policy*, 32:923–943., 2003.
- [Mas05] B Massey. Longitudinal analysis of long-timescale open source repository data. In *Proceedings of the 2005 Workshop on Predictor Models in Software Engineering, (St. Louis, Missouri, May 15 - 15, 2005)*. PROMISE '05, 2005.
- [MFH02] Audris Mockus, Roy T. Fielding, and James Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3), 2002.
- [Moz06] Mozilla Organization. Bugzilla information. <http://www.bugzilla.org/>, Mit Nov 29 15:28:03 CET 2006.
- [MV00] Audris Mockus and Lawrence G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance*, pages 120–130, October 2000.

- [Ope06a] Open Source Technology Group. Query Problem Reports. <http://freshmeat.net/screenshots/17301/18153/>, Fre Dez 29 15:26:05 CET 2006.
- [Ope06b] Open Source Technology Group. TkGnats-Household Projects. <http://www.timshel.ca/tkgnats/query.jpg>, Fre Dez 29 15:41:39 CET 2006.
- [Ope06c] Open Source Technology Group. Double Choco Latte. http://sourceforge.net/project/showfiles.php?group_id=1424, Mon Dez 18 14:23:27 CET 2006.
- [PHP06a] PHP Helpdesk. PHP Helpdesk. <http://phphelpdesk.sourceforge.net/help.html>, Mon Dez 18 12:30:53 CET 2006.
- [PHP06b] PHP Helpdesk. Screenshot PHP Helpdesk. <http://helpdesk.peachschools.org/index.php?login=goto>, Mon Dez 18 13:17:34 CET 2006.
- [php06c] phpBugTracker. phpBugTracker-Working with Bugs. <http://phpbt.sourceforge.net/docs/bugdetail.html>, Mit Dez 20 14:03:27 CET 2006.
- [php06d] phpBugTracker. Entering a bug. <http://phpbt.sourceforge.net/docs/userguide.html#PRIMER>, Mit Dez 20 15:50:32 CET 2006.
- [RBM03] G. Robles, J.M. Barahona, and M. Michlmayr. Evolution of volunteer participation in libre software projects: Evidence from debain. *ICSE '03 Workshop on Open Source Software Engineering, Portland, Oregon, May 3-10, 2003*.
- [RGB06] Gregorio Robles and Jesus M. Gonzalez-Barahona. Geographic location of developers at sourceforge. In *Proceedings of the 2006 international workshop on Mining software repositories, ACM Press*, pages 144–150, 2006.
- [SDD05] Katherine J. Stewart, David P. Darcy, and Sherae L. Daniel. Observations on patterns of development in open source software projects. In *5-WOSSE: Proceedings of the fifth workshop on Open source software engineering, St. Louis, Missouri, ACM Press*, pages 1–5, 2005.

- [SIA06] Sulayman K. Sowe, Stamelos Ioannis, and Lefteris Angelis. Identifying knowledge brokers that yield software engineering knowledge in oss projects. *Information and Software Technology*, 48(11):1025–1033, 2006.
- [SKS05] S.K. Sowe, A. Karoulis, and I. Stamelos. A constructivist view on knowledge management in open source virtual communities. In D.A. Figueiredo and A. Paula, editors, *Managing Learning in Virtual Settings: The Role of Context*, pages 290–308. Idea Group Inc., 2005.
- [SSA07] Sulayman K. Sowe, Ioannis Stamelos, and Lefteris Angelis. Understanding knowledge sharing activities in free/open source software projects: An empirical study. *Journal of Systems and Software*, 00:000–000, 2007.
- [VA 06a] VA Software Corporation. SourceForge[®] Features & Benefits. <http://dcl.sourceforge.net/>, Fre Dez 15 13:10:27 CET 2006.
- [VA 06b] VA Software Corporation. SourceForge, Optimizing Distributed Development. <http://www.vasoftware.com/sourceforge/>, Mit Dez 13 15:12:33 CET 2006.
- [VA 06c] VA Software Corporation. Collaboration. <http://www.vasoftware.com/sourceforge/collaboration.php>, Mit Dez 20 09:53:10 CET 2006.
- [VA 06d] VA Software Corporation. Interoperability. <http://www.vasoftware.com/sourceforge/interoperability.php>, Mit Dez 20 10:36:17 CET 2006.
- [VTG⁺06] S. Valverde, G. Theraulaz, J. Gautrais, V. Fourcassie, and R.V. Sole. Self-organization patterns in wasp and open source communities. *IEEE Intelligent Systems*, 21:36–40, 2006.
- [Wik06a] Wikipedia Organization. First Computer Bug. <http://upload.wikimedia.org/wikipedia/commons/8/8a/H96566k.jpg>, Mit Dez 6 11:57:51 CET 2006.
- [Wik06b] Wikipedia Organization. Bugzilla report. <http://en.wikipedia.org/wiki/Bugzilla>, Mit Nov 29 10:12:45 CET 2006.

- [Wik06c] Wikipedia Organization. Bugzilla Lifecycle. <http://en.wikipedia.org/wiki/Image:BzLifecycle.png>, Mit Nov 29 15:59:03 CET 2006.
- [Wik06d] Wikipedia Organization. SourceForge. <http://en.wikipedia.org/wiki/SourceForge>, Mon Dez 11 16:47:24 CET 2006.
- [Wik06e] Wikipedia Organization. Proprietary software. http://en.wikipedia.org/wiki/Proprietary_software, Mon Dez 11 16:53:11 CET 2006.
- [Wik06f] Wikipedia, the free encyclopedia. SourceForge. <http://www.reference.com/browse/wiki/SourceForge>, Mon Dez 11 17:26:25 CET 2006.
- [XGCM05] J. Xu, Y. Gao, S. Christley, and S. Madey. A topological analysis of the open source software development community. In *IEEE Proceedings of the 38th Hawaii International Conference on System Sciences, (IEEE, HICSS '05-Track 7), Jan 03-06, Big Island, Hawaii.*, page 198a., 2005.
- [ZW04] Thomas Zimmermann and Peter Weissgerber. Processing CVS data for fine-grained analysis. In *Proceedings of the International Workshop on Mining Software Repositories*, Edinburg, Scotland, UK, 2004.