



Free/Libre and Open Source Software Metrics

Sponsored through Framework Programme Sixth (Call 5) by



Document Information

Version: 2.0
Date : Oct 31, 2009
revision: 1

Owning Partner: WUW

Author(s):
Stefan Koch
Felipe Ortega
Liliana Tovar
Santiago Dueñas
Gregorio Robles

Reviewer(s):
Jesus M. Gonzalez – Barahona

To:
Public

Purpose of distribution:
Final version

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.

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: D4.1

Title:

Classification Report

License for distribution:

This work is licensed under a [Creative Commons Attribution-Share Alike 3.0 License](http://creativecommons.org/licenses/by-sa/3.0/)
(The license can be found in <http://creativecommons.org/licenses/by-sa/3.0/>)



Classification Report

Deliverable ID: D4.1

Page : 2 of 50

Version: 2.0

Date: Oct 31, 09

Status : Final


Confid : Public

Deliverable: D4.1

Title: Classification Report

Executive Summary:

This report details the results of the application of analytical techniques to the aggregated databases of the projects considered by FLOSSMetrics, and offers a classification schema for them. The classification schema will be used by the Melquiades web interface for classifying and supporting selection of projects. In part, this schema is also used in the “FLOSS Guide to SMEs” (deliverable D8.1.4) as a set of indicators about project activity, characteristics and community characterisation.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 3 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

CHANGE LOG

Ver.	Date	Author	Description
0.1	12/09/2008	Stefan Koch	Initial proposal for structure
0.2	01/10/2008	Gregorio Robles	Sections by URJC
0.3	10/10/2008	Liliana Tovar	Queries adapted
0.4	15/11/2008	Jesus M. Gonzalez – Barahona	General review
0.5	01/03/2009	Santiago Dueñas	Rework on simple and combined variables sections
0.6	15/03/2009	Santiago Dueñas	Complex metrics added
1.0	31/03/2009	Jesus M. Gonzalez – Barahona	Final review
1.5	08/07/2009	Stefan Koch	First round classification scheme
1.7	18/10/2009	Felipe Ortega	Second round classification scheme
2.0	31/10/2009	Jesus M. Gonzalez – Barahona	Review and final release

APPLICABLE DOCUMENT LIST

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




	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 4 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public


TABLE OF CONTENTS

1. Introduction	7
2. Simple first layer variables	9
2.1 Variables from Source Code Management Systems	9
2.1.1 SCM own variables	9
2.1.2 File variables	11
2.2 Variables from Mailing Lists	13
2.3 Variables from Tracking Systems	14
3. Combined Variables	16
3.1 Combined Variables based on Source Code Management Systems	16
3.2 Combined Variables based on Mailing Lists	20
3.3 Combined Variables based on Tracking Systems	21
4. Complex Variables	24
4.1 Evolution of first issue (bug) reports submitted by registered users	24
4.2 Evolution of first commits submitted by registered users	24
4.3 Evolution of new core members	25
4.4 Evolution of core members leaving the core team	25
4.5 Evolution of the balance in the core team	25
4.6 Average of committers longevity	25
4.7 Evolution of code contributors who submitted patches and changes	26
4.8 Total code contributors who submitted patches and changes	26
4.9 Evolution in number of events	26
4.10 Evolution in number of commits	26
4.11 Percentage of people working in old releases	27
4.12 Territoriality	27
4.13 Activity per committer	27
4.14 Number of lines per committer	27
5. Classification Scheme	28
5.1 Process Description	28
5.2 Classification Scheme	29
5.3 Second Round of Project Classification	33
5.3.1 EDA on FLOSSMETRICS Projects	34
5.3.2 Clustering analysis	36
5.3.3 Classification Trees	40
6. Appendixes	44
6.1 SQL Statements for complex variables	44
6.1.1 Evolution of first reports submitted by registered users	44
6.1.2 Evolution of fist commits submitted by registered users	44
6.1.3 Evolution of new core members	45
6.1.4 Evolution of core members leaving the core team	45

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 5 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

6.1.5	Evolution of the balance in the core team	46
6.1.6	Average of committers longevity	47
6.1.7	Evolution of code contributors who submitted patches and changes	47
6.1.8	Total code contributors who submitted patches and changes	47
6.1.9	Evolution in number of events	48
6.1.10	Evolution in number of commits	48
6.1.11	Percentage of people working in old releases	48
6.1.12	Territoriality	49
6.1.13	Activity per committer	49
6.1.14	Number of lines per committer	49
6.1.15	Evolution of first reports submitted by registered users	50

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	<p>Page : 6 of 50</p> <hr/> <p>Version: 2.0 Date: Oct 31, 09</p> <hr/> <p>Status : Final Confid : Public</p>
-----------------------------------------------------------------------------------	----------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 7 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public


1. INTRODUCTION

The objective of this report is to document the analysis performed on the data about projects in the FLOSSMetrics database, with the final aim of producing a classification schema and criteria for FLOSS projects. The variables mentioned in this report are also reflected in the database itself, which after the relevant revision contains specific tables to easily query about the criteria described here. Those criteria will be made available via the Melquiades¹ web service.

To produce this classification schema a set of variables were defined for each project in the database, starting from the “raw” data about project repositories to more complex aggregated information. These variables can be organized in three groups:

- **raw variables** or **simple variables** (first layer variables) are available in the data, directly as it is retrieved from software repositories and stored in the corresponding database.
- **combined variables** (second layer variables) are derived from simple variables, by combining or aggregating them. We define those variables by specifying the calculus to be performed with the raw data, such as mean or maximum values, values for specific periods, or values aggregated for a set of items.
- **complex variables** (third layer variables) are computed in a more complex fashion, based on statistical models or on computations spanning more than one repository. All the variables that are present in this deliverable have been included after examination of and discussion with the QUALOSS project. This project defined a group of metrics to measure the quality of a project and we decided to use those that measure the FLOSS communities, due to those being perfectly suitable for the purposes of FLOSSMetrics. In a later version of this deliverable and once the high level studies are completed, some other complex variables, derived from them, could be added to this list.

¹ <http://melquiades.flossmetrics.org>


	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 8 of 50
		Version: 2.0 Date: Oct 31, 09
		Status : Final Confid : Public

Applying a methodology based on these variables it will be possible to arrive at a classification scheme.

The classification scheme is based in some clustering techniques that serves to characterize and identify different subpopulations from a set of data (in our case, projects), according to several model descriptors. After this classification, an optimal classification tree can be defined, serving as a tool to classify new projects regarding the common traits shared with existing subpopulations identified in the previous set of projects.

The variables and scheme defined in this report will also be included and used in the SMEs guide to provide information and criteria that allow to costumers which FLOSS projects are more suitable for its needs.

The structure of this report follows an increasing abstraction level. In the next three sections, the simple, combined and complex variables available for the projects are defined. In the last section, the methodology and the classification scheme is presented.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 9 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

2. SIMPLE FIRST LAYER VARIABLES


This section provides a list and short description of the variables available in the first layer (raw) database. This amounts to providing an annotated list of variables readily available from the main tables in the databases for each kind of repository (commits in SCM system, messages in mailing lists, tickets in issue tracking system). It should be noted that (implicitly) the name of the project is also a part of the description (in the first level database there is in fact a database per each repository of each project), so every notion of uniqueness is dependent on combination with the project name.

2.1 VARIABLES FROM SOURCE CODE MANAGEMENT SYSTEMS

The main variables available for SCM repositories can be divided in two types: the variables from the SCM system itself and the variables from those files stored and managed by the repository.

2.1.1 SCM own variables


- **Repository URL.** Address of the analysed repository.
- **Author.** Developer of the changes in the source code. Most of the systems do not support this term and only store the information of the committer (see next point). GIT is the only system that supports this characteristic for which FLOSSMetrics offers data. The components of this variable are:
 - **Author id.**
 - **Author name.** Not provided in the anonymous version of the database or in public results.
 - **Author email.** Not provided in the anonymous version of the database or in public results.

	<p style="text-align: center;">Classification Report</p> <p style="text-align: center;">Deliverable ID: D4.1</p>	Page : 10 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

- **Committer.** Person responsible of apply the set of modifications or changes (a commit), submitted by the author or the developer, on the repository. This person has write access to the repository and might or not be the author of the changes. As in the case of the author and depending on the characteristics of the analysed repository some of the next components are not always available:
 - **Committer id.** Identifier of the committer in the repository. Present in all the supported repositories.
 - **Committer name.** Only for GIT repositories. Not provided in the anonymous version of the database or in public results.
 - **Committer email.** Only for GIT repositories. Not provided in the anonymous version of the database or in public results.

- **Commit date.** Date when the changes were applied in the repository.
- **Commit message.** Message that describes the objective or the motivation of the changes.
- **Revision file identifier.** Identifies the status of a file at a specific point in time. The status of a file is the set of name, path, branch and content that the file has at a certain instant of time.
- **Revision repository identifier.** Identifies the status of the repository at a specific point in time. The status of a repository is a repository snapshot with a set of files, which their own status, at a certain instant of time.
- **Action type.** Type of the changed applied to a file on a commit. The type can be one of the next, but not all the repositories support the full list:
 - *Added* ('A'): the file has been added to the repository.
 - *Modified* ('M'): the file has been modified in its content.
 - *Deleted* ('D'): the file has been deleted.
 - *Moved* ('V'): the file has been moved to another or to the same path keeping or not its name.
 - *Copied* ('C'): the file has been copied from another existing file in the repository. It is similar to add a file to the repository.
 - *Replaced* ('R'): the file has been replaced. This means that an existing file was removed and a new one with the same name has been added.

- **Tag.** Label for identifying a snapshot of the repository at a specific point in time.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 11 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

- **Branch name.** Name of a line of development. These lines are independent but can share the status of a set of files between them. The main development line is usually named 'trunk'.

2.1.2 File variables


The next variables are available for every file of the analysed repository:

- **File name.** Name of a file at a specific instant of time.
- **File path.** Path of a file at a specific instant of time.
- **File branch.** Branch in which a file can be found at a specific point in time.
- **File extension.** Extension of a file, if any.
- **File type.** Type of a file according to its extension and properties. The types can be one of the following:
 - *build* : files for building the source code such as autotools or make files.
 - *devel-doc* : specific files with relevant information for the developer.
 - *documentation*: files of documentation as manuals, help files, tutorials, etc.
 - *image*: image files with distinct formats such as BMP, GIF, JPG, PNG, SVG, TIFF, and so on.
 - *i18n* : files for translation and localization
 - *multimedia* : multimedia contents such as sound or video files. For instance AVI, MPEG, MP3, Midi, OGG or WAV files.
 - *package* : files that compress or package other files. These files might be ZIP or TAR files; Debian packages; RPM packages and so on.
 - *source* : source code files.
 - *ui* : files for the user interface
 - *unknown*: for those files that its type could not be matched with any of the previous types.

The following variables are only available for those files that contain source code and are related with the revision of each file.

- **File language.** Programming language of a source code file. The possible languages are those that SLOCCcount tool² supports. This list contains almost 30 languages including C, C++, C#,

²<http://www.dwheeler.com/sloccount/>

	<p style="text-align: center;">Classification Report</p> <p style="text-align: center;">Deliverable ID: D4.1</p>	Page : 12 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

Java, Perl, Ada, Python or Assembly.


- **File LOC.** Number of lines of code (LOC) of a file in a specific revision. A line of code is any line of program text, including comment lines, blank lines, etc.
- **File SLOC.** Number of physical lines of code (SLOC) of a file in a specific revision. The definition of SLOC is the same used by SLOCCount: *“a physical source line of code (SLOC) is a line ending in a newline or end-of-file marker, and which contains at least one non-whitespace non-comment character. Comment delimiters (characters other than newlines starting and ending a comment) are considered comment characters. Data lines only including whitespace (e.g., lines with only tabs and spaces in multiline strings) are not included”*³.

The next variables are only given for those files with one of the following programming languages: C, C++, Java, Perl and Python. In addition these ones are specific of the file revision.

- **Blank lines of a file.** Number of blank lines including lines with white spaces and tabs.
- **Comment lines of a file.** Number of lines with comments on a file.
- **Comments of a file.** Number of comments the file contains.
- **Functions of a file.** The number of functions the specific file contains.
- **McCabe's cyclomatic complexity of a file.** The definition of this metric is the given by McCabe⁴. McCabe's complexity is defined at function level but the database only stores information of files. Therefore, instead of store cyclomatic complexity values per function, some other variables were defined with the intention of lose the less information. These variables are the following:
 - **Maximal McCabe complexity of a file.** The maximal cyclomatic complexity found in the functions of a file.
 - **Minimal McCabe complexity of a file.** The minimal cyclomatic complexity found in the functions of a file.
 - **Sum of McCabe complexity of a file.** The sum of all the cyclomatic complexity values obtained from a file.
 - **Mean McCabe complexity of a file.** The mean of the cyclomatic complexity values obtained from a file.

³<http://www.dwheeler.com/sloccount/sloccount.html>

⁴ Thomas J. McCabe. A complexity measure. *IEEE Transactions of Software Engineering*, SE-2(4):308-320, 1976

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 13 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public


- **Median McCabe complexity of a file.** The median of the cyclomatic complexity values obtained from a file.
- **Halstead's software science metrics of a file⁵.** The definitions of these metrics are the given by Kan. These variables are the next:
 - **Halstead length.** Variable that includes the total number of operator occurrences and total number of operand occurrences in a file.
 - **Halstead volume.** This is a Halstead metric that contains the minimum number of bits required for coding the file.
 - **Halstead level.** Level at which the source code of a file can be understood.
 - **Halstead difficulty.** Level of difficulty in the file.

2.2 VARIABLES FROM MAILING LISTS

The following variables are available for mailing lists repositories:

- **Mailing list address.** Address of the mailing list.
- **User.** Person which her email address appears, as a sender or as one of the receivers, in any of the emails sent to the mailing list. The components defined for users are the following:
 - **User id.** Identifier of the user.
 - **User name.** This name is the user's nick or her real name, if was given. Not available in the anonymous database or in public results.
 - **User email address.** Not available in the anonymous database or in public results.
- **Message identifier.** Hash that identifies the message as unique from other messages.
- **Message subject.** The textual subject of the email message.
- **Message body.** The body or content of the email message.
- **Message date.** Local time and date when the client sent the message to the mailing list. From the date of the message two components are defined:
 - **Message date.** The local time and the date of the message itself.
 - **Message date time-zone.** The region where the client was sending the message.

⁵ Stephen H. Kan. *Metrics and Models in Software Engineering (2nd Edition)*. Addison-Wesley Professional, September 2003


	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 14 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

- **Mailing list date.** This date refers to the local time and date when the server, that manages the mailing list, deliver the message to its users. Like the message date, two components are defined:
 - **Mailing list date.** The local time and the date of the deliver itself.
 - **Mailing list time-zone.** The region where the server was delivering the message.
- **Message poster.** Identifier of the person who sent the message.
- **Message receiver.** Identifier of the entity (mailing list or person) who received the message.
- **Message response.** Identifier of the message that is a response of other one.

2.3 VARIABLES FROM TRACKING SYSTEMS


This section describes all the variables defined for tracker systems. The list is the following:

- **Tracker URL.** Address of the analysed tracker.
- **Contributor identifier.** Unique identifier of a person in the system. The user can have different roles such as reporter or fixer and so on.
- **Submitter.** Contributor that sent the report to system.
- **Summary of the report.** Brief description of the report
- **Report description.** Textual description of the report.
- **Report date.** Date when the report was notified to the system.
- **Report category.** Type of the report.
- **Report status.** Status of the report at a certain instant of time. It can take one of the following values:
 - *open* : new report without assignation
 - *closed* : fixed report
 - *deleted* : invalid report
 - *pending* : assigned report but not fixed
- **Report priority.** Urgency of the report at a specific point in time. Usually this field is modified by the tracker-master as users do not have sufficient knowledge on the software to know the correct value.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 15 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

It can take following values (from high priority to lower one): “immediate”, “urgent”, “high”, “normal” and “low”.

- **Report severity.** Severity of the report at specific point in time. How this report affects the use and development of the software. Possible values are (from high severity to lower one): “blocker”, “critical”, “major”, “normal”, “minor”, “trivial” and “enhancement”.
- **Assignment.** Developer in charge of fixing the report at a specific point in time.
- **Date of the comment.** Date when a report comment was posted.
- **Body of the comment.** Textual description of a report comment.
- **Change type.** Value change of any of the variables of a report at a certain point in time.
- **Date of the change.** Date when the change of a variable was performed.
- **Change description.** Textual description of a change.
- **Name of the attachment.** Name of the element attached to the report.
- **Description of the attachment.** Textual description of the element attached to the report.
- **URL of the attachment.** Address of the element attached to the report.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 16 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

3. COMBINED VARIABLES

Here comes a list of variables aggregated and computed from the simple level variables (e.g. commits per month, number of distinct committers per month etc.). These are computed based on rather simple rules as described below. As a very general rule, computing these should be possible using a (possibly elaborate) SQL-statement. Some of these statements can be found in Melquiades documentation pages⁶.

More complex measures are defined in the following section.

3.1 COMBINED VARIABLES BASED ON SOURCE CODE MANAGEMENT SYSTEMS


- **Commits**

- Total number of commits
- Number of commits per unit of time
- Aggregated number of commits over time
- Maximum and minimum number of commits per unit of time
- Mean and median of commits per unit of time

- **Actions**

- Total number of actions
- Total number of actions per type
- Number of actions per unit of time
- Aggregated number of actions over time
- Maximum and minimum number of actions per unit of time
- Mean and median of actions per unit of time
- Number of actions per unit of time and per type
- Aggregated number of actions per type over time
- Maximum and minimum number of actions per unit of time and per type
- Mean and median of actions per unit of type and per type

⁶ <http://melquiades.flossmetrics.org/wiki/>

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 17 of 50
		Version: 2.0 Date: Oct 31, 09
		Status : Final Confid : Public

- **Committers / Authors**


- Total number of committers/authors
- Number of distinct committers/authors per unit of time
- Aggregated number of distinct committers/authors over time
- Maximum and minimum number of distinct committers/authors per unit of time
- Mean and median of distinct committers/authors per unit of time
- Number of new committers/authors per unit of time
- Aggregated number of new committers/authors over time
- Maximum and minimum number of new committers/authors per unit of time
- Mean and median of new committers/authors per unit of time

- **Commits – Committers/Authors**

- Number of commits per committer/author and per unit of time
- Aggregated number of commits per committer/author over time
- Maximum and minimum number of commits per committer/author and per unit of time
- Mean and median of commits per committer/author and per unit of time
- Maximum and minimum number of commits per committer/author
- Mean and median of commits per committer/author

- **Actions – Committers/Authors**

- Number of actions per committer/author and per unit of time
- Aggregated number of actions per committer/author over time
- Maximum and minimum number of actions per committer/author and per unit of time
- Mean and median of actions per committer/author and per unit of time
- Maximum and minimum number of actions per committer/author
- Mean and median of actions per committer/author
- Number of actions per committer/author, type and unit of time
- Aggregated number of actions per committer/author, type over time
- Maximum and minimum number of actions per committer/author, type and unit of time
- Mean and median of actions per committer/author, type and unit of time
- Maximum and minimum number of actions per committer/author and per type
- Mean and median of actions per committer/author and per type

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 18 of 50
		Version: 2.0 Date: Oct 31, 09
		Status : Final Confid : Public

- **Files**


- Total number of files in the whole history of the project
- Number of files per unit of time
- Maximum and minimum number of active (not deleted) files
- Mean and median of active (not deleted) files per unit of time
- Total number of files per type in the whole history of the project
- Number of files per type and per unit of time
- Aggregated number of files per type over time
- Maximum and minimum number of active (not deleted) files per type
- Mean and median of active (not deleted) files per type
- Number of files per language
- Accumulated number of files per language

- **Commits – Files**

- Total number of commits per file
- Number of commits per file and per unit of time
- Accumulated number of commits per file over time
- Maximum and minimum number of commits over files per unit of time
- Mean and median of commits over files per unit of time

- **Actions – Files**

- Total number of actions per file
- Number of actions per file and per unit of time
- Accumulated number of actions per file over time
- Maximum and minimum number of actions over files per unit of time
- Mean and median of actions over files per unit of time
- Total number of actions per type and per file
- Number of actions per type, file and unit of time
- Accumulated number of actions per type, file over time
- Maximum and minimum number of actions over files per type and per unit of time
- Mean and median of actions over files per type and per unit of time

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 19 of 50
		Version: 2.0 Date: Oct 31, 09
		Status : Final Confid : Public

- **Size variables (LOC / SLOC)**


- Total LOC/SLOC of the project in the whole history of the project
- Project size in LOC/SLOC per unit of time
- Maximum and minimum size in LOC/SLOC per unit of time
- Mean and median of size in LOC/SLOC per unit of time
- LOC/SLOC added per unit of time
- LOC/SLOC deleted per unit of time
- Maximum and minimum LOC/SLOC added per unit of time
- Mean and median LOC/SLOC deleted per unit of time

- **Files – Size variables**

- Maximum and minimum size in LOC/SLOC per file
- Mean and median of size in LOC/SLOC per file
- Maximum and minimum of size in LOC/SLOC per file
- Mean and median of size in LOC/SLOC per unit of time
- Maximum and minimum file sizes in LOC/SLOC per unit of time
- Mean and median of file sizes in LOC/SLOC per unit of time
- LOC/SLOC added per file and per unit of time
- LOC/SLOC deleted per file and per unit of time
- Maximum and minimum file sizes in LOC/SLOC per unit of time
- Mean and median sizes in LOC/SLOC per unit of time

- **Files – Committers/Authors**

- Number of distinct committers/authors per file
- Number of distinct committers/authors per file
- Maximum and minimum number of committers/authors per file
- Mean and median of committers/authors per file
- Number of files per committers/authors and per file type

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 20 of 50
		Version: 2.0 Date: Oct 31, 09
		Status : Final Confid : Public

3.2 COMBINED VARIABLES BASED ON MAILING LISTS

- **Messages or posts**


- Total number of messages
- Number of messages per unit of time
- Aggregated number of messages over time
- Maximum and minimum number of messages per unit of time
- Mean and median of messages per unit of time
- Maximum and minimum of messages size
- Mean and median of messages size

- **Threads**

- Total number of threads
- Size of the thread per unit of time (defining size as the total number of replies to an initial posting)
- Aggregated size per thread over time
- Maximum and minimum size of threads
- Mean and median size of threads
- Maximum and minimum message thread depth, defined as the number of replies to a reply, to a reply, etc, to an initial posting.
- Mean and median of message thread depth

- **Authors**

- Total number of distinct authors
- Number of distinct authors per unit of time
- Aggregated number of distinct authors over time
- Maximum and minimum number of authors per unit of time
- Mean and median of authors per unit of time
- Number of new authors per unit of time
- Aggregated number of new authors over time
- Maximum and minimum number of new authors per unit of time

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 21 of 50
		Version: 2.0 Date: Oct 31, 09
		Status : Final Confid : Public

- Mean and median of new authors per unit of time

- **Messages – Authors**

- Total number of messages per author
- Number of messages per author and per unit of time
- Aggregated number of messages per authors over time
- Maximum and minimum number of messages per author over all authors
- Mean and median of messages per author over all authors
- Total of number of replies per author
- Number of replies per author and per unit of time
- Aggregated number of replies per author over time
- Maximum and minimum number of replies per author over all authors
- Mean and median of replies per author over all authors


- **Authors – Threads**

- Total number of distinct authors per thread
- Maximum and minimum number of distinct authors per thread
- Mean and median of distinct authors per thread
- Number of messages per author and per thread
- Maximum and minimum number of messages per author and per thread
- Mean and median of messages per author and per thread

3.3 COMBINED VARIABLES BASED ON TRACKING SYSTEMS

- **Reports**

- Total number of reports
- Number of reports per unit of time
- Aggregated number of reports over time
- Maximum and minimum number of reports per unit of time
- Mean and median of reports per unit of time
- Total number of reports per category
- Number of reports per category and per unit of time

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 22 of 50
		Version: 2.0 Date: Oct 31, 09
		Status : Final Confid : Public

- Aggregated number of reports per category over time
- Maximum and minimum number of reports per category and per unit of time
- Mean and median of reports per category and per unit of time
- Number of reports per category and per unit of time
- Aggregated number of reports per category over time
- Maximum and minimum number of reports per category and per unit of time
- Mean and median of reports per category and per unit of time
- Number of reports per status and per unit of time
- Aggregated number of reports per status over time
- Maximum and minimum number of reports per status and per unit of time
- Mean and median of reports per status and per unit of time
- Maximum, minimum, mean and median defect resolution time, defined as the time difference between the date a bug was opened and the date it was closed (or confirmed).⁷


- **Contributors**

- Total number of contributors
- Number of contributors per unit of time
- Aggregated number of contributors over time
- Maximum and minimum number of contributors per unit of time
- Mean and median of contributors per unit of time
- Number of new contributors per unit of time
- Aggregated number of new contributors over time
- Maximum and minimum number of new contributors per unit of time
- Mean and median of new contributors per unit of time


- **Reports – Contributors**

- Number of reports per contributor per unit of time
- Aggregated number of reports per contributor over time
- Maximum and minimum number of reports per contributor and per unit of time
- Mean and median of reports per contributor and per unit of time
- Number of reports per contributor, status and unit of time
- Aggregated number of reports per contributor, status over time

⁷ Several alternative definitions are possible here. It should also be noted that bugs that were opened and closed and then re-opened should be treated separately.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	<p>Page : 23 of 50</p> <hr/> <p>Version: 2.0 Date: Oct 31, 09</p> <hr/> <p>Status : Final Confid : Public</p>
-----------------------------------------------------------------------------------	----------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------

- Maximum and minimum number of reports per contributor, status and unit of time
- Mean and median of reports per contributor, status and unit of time

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 24 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

4. COMPLEX VARIABLES

This section describes a set of variables obtained by combining those described on the previous sections: simple and combined variables; and those from the distinct sources: Source Code Management Systems, mailing lists and tracking systems.

All the variables have a description and a group of operations and/or SQL statements used to compute them from the FLOSSMetrics database.


Most of the variables in this section were defined in the QUALOSS project, specifically the community variables, and are later used by FLOSSMetrics for classifying projects. After the results and learned lessons from the High Level Studies (WP5) are completed, some other variables could be identified and included in this section. This includes, for example, efficiency scores derived from a data envelopment analysis.

4.1 EVOLUTION OF FIRST ISSUE (BUG) REPORTS SUBMITTED BY REGISTERED USERS

- **Description:** Retrieving the date of the first bug for each member of the community, we are able to know if the number of new members reporting bugs remains stable.
- **Methodology:** Measure monthly the first bug submitted by registered people, taking into account the slope of the resultant line ($y=mx+b$) while measuring the aggregated number and periods of one year.

4.2 EVOLUTION OF FIRST COMMITS SUBMITTED BY REGISTERED USERS

- **Description:** Retrieving the date of the first commit for each member of the community, it is possible to determine whether or not the number of new members committing remains stable
- **Methodology:** First commit of each detected committer in the SCM is retrieved and its monthly rate calculated, taking into account the slope of the resultant line ($y=mx+b$) while measuring the aggregated number and periods of one year:

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 25 of 50
		Version: 2.0 Date: Oct 31, 09
		Status : Final Confid : Public

4.3 EVOLUTION OF NEW CORE MEMBERS

- **Description:** Retrieving this information gives an estimator of how the core of contributors is evolving. Thus, it can be seen whether there is a natural regeneration of core developers.
- **Methodology:** Checking who forms the core team of developers (those with the 80% of the commits) and then analysing the first commit and monthly results of each new member who starts working on the core group, we can get the regeneration of core developers. Note: the results are obtained taking into account the slope of the resultant line ($y=mx+b$) while measuring the aggregated number and periods of one year:

4.4 EVOLUTION OF CORE MEMBERS LEAVING THE CORE TEAM


- **Description:** Taking into account this variable we can estimate if there is a dramatic decrease in the number of core developers, and so, a risk in the regeneration.
- **Methodology:** First who forms the core team of developers (those with the 80% of the commits) is calculated. After this, any number of people who disappear, does not commit in the next months, from this core team is counted as one.

4.5 EVOLUTION OF THE BALANCE IN THE CORE TEAM

- **Description:** Number of people who left the core team minus number of new members of the core team monthly.
- **Methodology:** Calculate the evolution of new core members and the evolution of member that leaves the core. Then, calculate the difference between these two variables.

4.6 AVERAGE OF COMMITTERS LONGEVITY

- **Description:** Average age of people working on a project. This metric is focused on the average of years worked by each developer. With this approximation, we are able to know if members are approaching this limit and we can estimate future effort needs.
- **Methodology:** Measure the number of months in which a committer appears. This is her age in the project. The average of all of them is the longevity of a committer in the community.
 - *Black:* the longevity is younger than 1 year.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 26 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

4.7 EVOLUTION OF CODE CONTRIBUTORS WHO SUBMITTED PATCHES AND CHANGES

- **Description:** Evolution of people who contribute to the source code and reporting bugs. A way to retrieve this data is to analyse those committers and reporters with the same nickname.
- **Methodology:** Count the number of people committing changes in a major release, Taking into account the slope of the resultant line ($y=mx+b$) while measuring the aggregated number and periods of one year:

4.8 TOTAL CODE CONTRIBUTORS WHO SUBMITTED PATCHES AND CHANGES


- **Description:** Number of people involved in the project being a committer and also reporting bugs. With this variable we can measure the size of a community.
- **Methodology:** Same metric than above but this is the sum of all of them and not the evolution.

4.9 EVOLUTION IN NUMBER OF EVENTS

- **Description:** An event is defined as any kind of activity measurable from a community. Generally speaking, posts, commits or bug reports. Monthly analysis will provide a general view of the project and its tendency.
- **Methodology:** Total number of commits, posts and bug reports. The indicator will be measured taking into account the slope of the resultant line ($y=mx+b$) while measuring the aggregated number and periods of one year.

4.10 EVOLUTION IN NUMBER OF COMMITS

- **Description:** Monthly analysis will provide a general view of the project. In this way an increase or decrease in the number of commits will show the tendency of the community.
- **Methodology:** Total number of commits per month taking into account the slope of the resultant line ($y=mx+b$) while measuring the aggregated number and periods of one year.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 27 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

4.11 PERCENTAGE OF PEOPLE WORKING IN OLD RELEASES

- **Description:** Number of people working on old releases out of total work on the project. We can determine how supported are the old releases for maintenance purposes.
- **Methodology:** Number of people working on old releases divided by the total amount of people active people working nowadays on the project.

4.12 TERRITORIALITY


- **Description:** This metric shows the territoriality in a project. Generally speaking, most of the files are touched or handled by just one committer. It means that high levels of orphaning may be seen as a risk situation. If a developer leaves the project, her knowledge will disappear and all her files are totally unknown by the rest of the developers team.
- **Methodology:** Percentage of files worked just for one committer. If this number is really high, it means that there are no big sub-groups of people.

4.13 ACTIVITY PER COMMITTER

- **Description:** High number of SLOC, e-mails or bugs to be fixed per active developer may mean that they are overworked. In this case, the community is clearly busy and they need more people to help on it.
- **Methodology:** Measured counting the number of people working on the project, out of number of people working on the whole project and taking into account the whole set of activities to carry on.

4.14 NUMBER OF LINES PER COMMITTER

- **Description:** Relationship between committers and total number of lines or files. With this absolute number, we are able to check the number of lines per committer. Thus, just regarding to the source code, we can say if they need more resources on it.
- **Methodology:** Number of lines per committer, taking into account the number of active committer, and the total size in SLOC of the project.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 28 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public


5. CLASSIFICATION SCHEME

This section will describe, according with the variables defined in the previous sections, a methodology and a scheme for the classification of FLOSS projects. The motivation for this is to enable any user of FLOSSMetrics, both users/companies as well as researchers, to be quickly able to distinguish a limited number of different project types. Resulting types could be small/dormant or large and active. This is supposed to act as a very quick indicator for scanning potential candidates for adoption or inclusion into datasets. The second interesting result of applying classification is that it will allow an overview of the overall FLOSS population, and the populations of it's sub-groups. The number of projects in each group could also be tracked over time, to give an indication of whether the number of successful projects (of course depending on definition) is increasing or not.

5.1 PROCESS DESCRIPTION

We decided to set up on ongoing, automated process for discovering a classification scheme within the data. This is based on doing a clustering analyses on the projects within the FLOSSMetrics database, and is implemented such that it can be started all over again at later points in time (the code in the free statistical language and environment R is submitted). The following steps are necessary to arrive at a classification scheme:

1. Get the necessary variables from the FLOSSMetrics database for all projects. These will necessarily all be on project level. Prime candidates are number of developers, size, total number of bug reports and similar. With regard to this report, these will mostly be combined and complex variables, with a few simple exceptions like the project age.
2. Apply a clustering algorithm. Based on this aggregated data, a k-means clustering algorithm will return a number of clusters previously determined. For now, we set a target of 4 clusters. As a main benefit of this classification will be easy filtering by end users, a larger number might prove to be problematic and confusing. The clustering algorithm will return a set of clusters together with the attribute values of the centers of the clusters, number of

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 29 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

members, and the assigned cluster for each project. The assigned cluster can then be saved back into the FLOSSMetrics database for publishing it on the website.

3. For an easier interpretation and an insight into which attributes are most important for distinguishing between the groups, an additional step could be to derive an automated regression tree on the same data. In this, the assigned cluster is determined as the goal, and a tree is derived to most closely arrive at this solution. In this tree, each node contains a single attribute out of the total set together with borders to distinguish between the children. This could also be used by other interested parties like companies to place their own projects without having to donate their full data.

5.2 CLASSIFICATION SCHEME

As defined above, the following classification scheme is the first one derived from the data, and therefore not a fixed results. In addition, it is currently based on a small number of projects, and thus might not be stable yet. We will show the different results from each step:

1. At the current date (March 2009), and due to the switch to CVSAAnaly2, only a limited set of projects is available. As we tried to include all different input types (i.e. programmers, posters and mailers), we only have 36 projects left. This number is expected to increase considerably in the next time (and would be much higher if the CVSAAnaly1 projects would have been included as well, but for the sake of consistency, those were omitted). Using queries to the FLOSSMetrics database, the different variables were retrieved from each of the three sources (source code control, issue tracking and mailing lists), and were merged based on project identification. This resulted in a file for the 36 projects, with the following variables included in this preliminary study:

- Year the project was established
- Number of committers
- Number of bug reporters
- Number of posters
- Number of commits
- Number of codefiles
- Number of bugs
- Number of messages

2. A clustering set at a 4 cluster solution derived the following clusters:


K-means clustering with 4 clusters of sizes 12, 13, 7, 4

Cluster means:

	Year	number_developers	number_commits	number_codefiles	NumberofBugs
1	2002.417	21.33333	615.7500	1091.5000	138.1667
2	2002.615	16.76923	398.5385	404.2308	413.1538
3	2001.429	18.71429	657.1429	804.1429	2308.4286
4	2003.750	22.75000	475.2500	2794.5000	52.7500

	CountBugsPeople	CountMessages	CountMsg_People
1	46.08333	35.83333	2.500000
2	137.76923	49.00000	3.230769
3	769.42857	26.00000	2.857143
4	17.50000	8.25000	3.250000


Also, the cluster membership of each project is returned and saved back to the database. For an interpretation, it is necessary to take a look at the centers: We find quickly that the year is quite similar for all clusters, and therefore the distinction in old/young does not play a major role in the data. Also the number of developers is not a major factor, but together with commits and codefiles we see that clusters 1 and 4 seem to contain larger projects. Cluster 3 seems to focus on bugs, which shows in high values for bug reports and also reporters. This could actually be taken as a sign for high-quality projects (but of course additional metrics should be included for that). Cluster 2 in general, except for mail related metrics, seems to contain mostly smaller and less active projects. It could be hypothesized that these projects are in an earlier stage of development. It is interesting to see that this cluster is actually smallest in numbers.

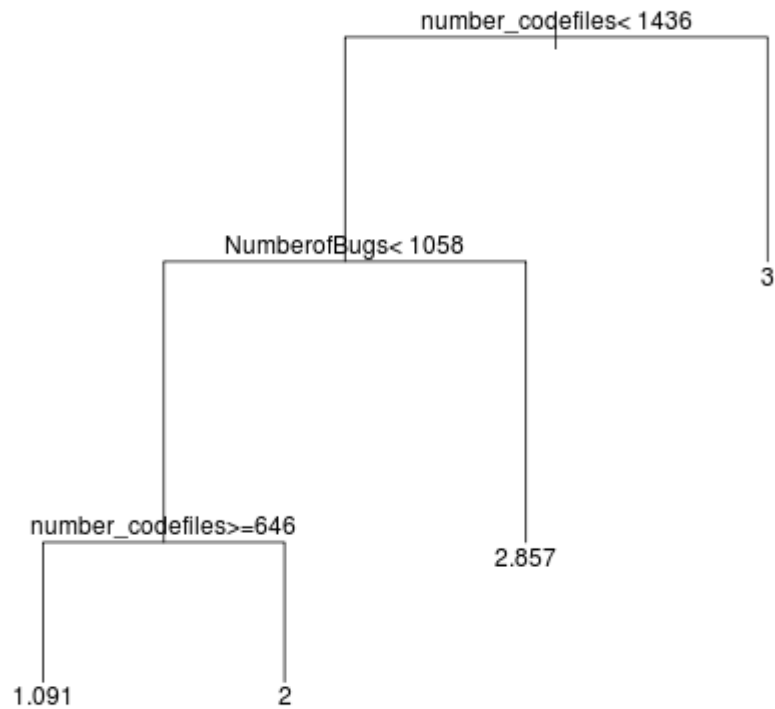
	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 31 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

3. For taking a look at which attributes are most important, and how a new project would be classified without restarting the complete clustering, a regression tree is generated with cluster membership as target attribute, and again using the other variables as predictors. This returns the following results:


- 1) root 36 34.7500000 2.083333
- 2) number_codefiles < 1435.5 29 15.4482800 1.862069
- 4) NumberofBugs < 1057.5 22 5.4545450 1.545455
- 8) number_codefiles >= 646 11 0.9090909 1.090909 *
- 9) number_codefiles < 646 11 0.0000000 2.000000 *
- 5) NumberofBugs >= 1057.5 7 0.8571429 2.857143 *
- 3) number_codefiles >= 1435.5 7 12.0000000 3.000000 *

This can also be represented graphically, and clearly shows the importance of codefiles and bugs as distinguishing features between the different clusters:

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 32 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public



Summing up, this has shown that a classification scheme can quite easily be established. It seems more prudent to do this data-driven, and not theory-driven. A classification scheme will have several positive side effects in helping users of the FLOSSMetrics database, as well as providing interesting research stimulus, and long-term tracking of cluster sizes to characterize the FLOSS population and the related change.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 33 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public


5.3 SECOND ROUND OF PROJECT CLASSIFICATION

After the initial attempt to outline a project classification scheme (describe in the past 2 subsections), we undertake here a more in-depth analysis of this aspect. On one side, the number of projects analysed in the FLOSSMetrics platform has grown substantially, from the previous set that was utilized for the first round. On the other side, this larger set will let us gain significance level in the statistical analyses applied to obtain the new classification scheme, and GNU R offers now a new set of techniques better aimed to tackle this type of problems.

The procedure adopted for this second round is as follows:

- We perform an initial phase of *Exploratory Data Analysis*, to identify existing range of values in available data. In this way, we can evaluate whether it should be convenient to apply any kind of transformation on the data set to improve the efficiency of subsequent statistical techniques applied on it.
- After that, we apply 2 different *clustering techniques* to characterize the distinct subpopulations (if any) that can be identified in this data set, according to several covariates that can be considered as descriptors for our model.
- Finally, we analyse and compute the optimal *classification tree*, aimed to serve as a tool to classify new projects regarding the common traits shared with existing subpopulations identified in our initial set of projects.

Therefore, the current analysis constitutes a new approximation to the problem of characterizing the best parameters for classification of FLOSSMetrics projects, following a coherent taxonomy. In this regard, as we will see the larger size of the new sample provides a valuable benefit for improving the significance of our analysis. The new results conclude that there is little correlation between the longevity of FLOSS development projects and most of its descriptive parameters, except for the total number of commits exhibited. This constitutes the only one parameter that may offer a hint about the most probable working time-frame of the project under analysis.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 34 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

5.3.1 EDA on FLOSSMETRICS Projects

For each project included in the FLOSSMetrics platform, we have computed several distinctive covariates that could serve to characterize them, as for the classification and clustering analysis that we are performing. The following covariates have been considered to describe each project:

- *months*: Duration of the project, in months (considered as the time-frame for which we have data available in the corresponding SCM repository).
- *ncommitters*: Number of different committers in the project.
- *ncommits*: Number of commits performed in the project history.
- *nfiles*: Number of different source files stored in the SCM repository.
- *nreporters*: Number of unique reporters of bugs in the project.
- *nreports*: Number of bug reports sent to the BTS of the project.

In this analysis, we have filtered out $n = 113$ projects from the initial set of analyzed projects available in FLOSSMetrics, corresponding to those sampled projects containing valid values for every covariate mentioned in the previous list. The criterion for selecting these project has been to choose those ones with available values for all the parameters selected above. In particular, we had to filter out parameters coming from the analysis of mailing lists, since the number of projects with available information for all sources under analysis in FLOSSMetrics (CMS, mailing lists and BTS) was too low. As a result, the followed criterion is the one maximizing the number of projects providing information for the analysis at this time.

Figure 5.1 shows the *box and whisker* plot summarizing the range of values obtain for each covariate y every project analysed. As we can see, we have a clear issue regarding quite distinct range of values for the different covariates considered in the analysis. In the case of *ncommits* and *nfiles*, the number of outliers in our sample is clearly disproportionated, and we may run the risk of overwhelming the rest of results with such highly extreme values.

In this cases, the usual solution is to apply a suitable transformation to the covariates, so that we can ensure that all ranges of values are comparable, to a reasonable extent. Therefore, we decide to apply a *logarithmic* transformation, taking the $\log_{10}()$ value for each covariate. As we can see in Figure 5.2, the transformation allows us to obtain ranges of values that can be better managed, with many less outliers for every covariate. We still can appreciate some outliers in 3 of the covariates (*ncommitters*, *nreporters* and *nreports*), though they present a very short departure from the main body of values in each of these parameters.

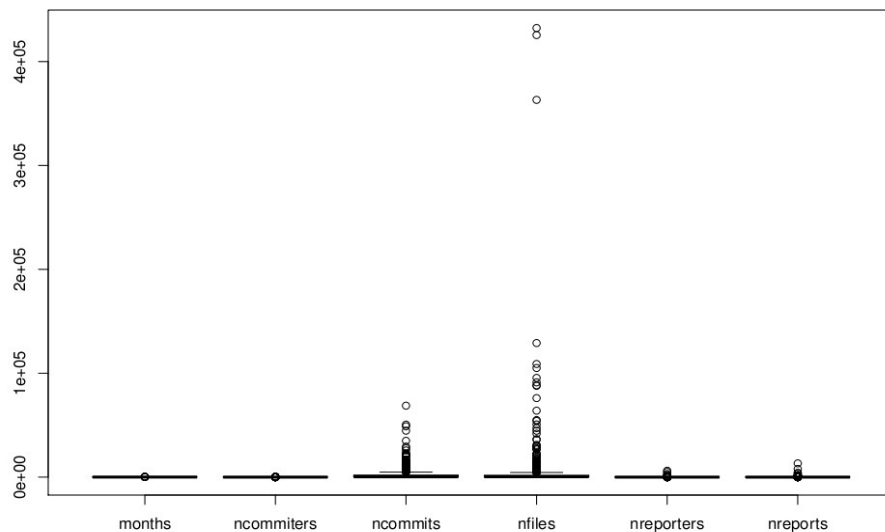


Figure 5.1 Box and whisker plot for covariates included in our analysis

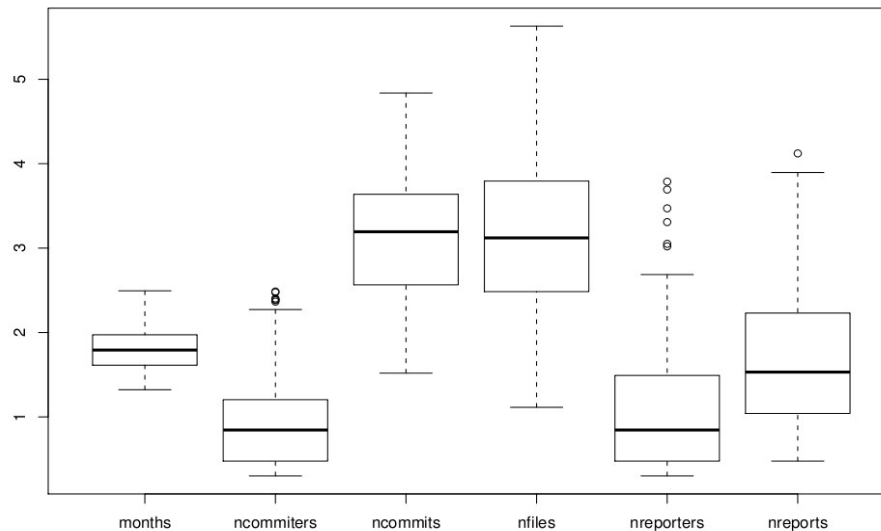


Figure 5.2 Box and whisker plot of our covariates, after the application of the log₁₀() transformation.

Once we have prepared our data for the analysis, we can continue with the clustering analysis and classification tree.

5.3.2 Clustering analysis

Clustering analysis is a term that encompasses several numerical methods aimed to identify clusters or groups, each one composed by a set of data samples sharing homogeneous values of some of the covariates considered in the model, and separated from the other groups. This is an attempt to offer a formal method to implement classification methods that can be performed with a naked eye by human observers in 2D and 3D representations. Further details of these kind of methods can be found in Everit *et al.* (2001) and Gordon (1999). We will apply here two different type of techniques: *k-means clustering* and *model-based clustering*.

K-means tries to find clusters of observations (a given number of clusters must be provided beforehand), minimising the overall within-group sum of squares over all variables considered in the analysis. The most difficult part is trying to decide the optimal number of clusters for the trial,

since there is no general recommendation valid for all possible cases. In our analysis, after several trials it seemed that $n=3$ clusters offered a reasonable set of differentiated values for each subgroup, provided in the following output:

K-means clustering with 3 clusters of sizes 48, 17, 48

Cluster means:

	months	ncommiters	ncommits	nfiles	nreporters	nreports
1	1.665301	0.6211936	2.629384	2.463306	0.6265206	1.164275
2	1.961759	2.0536210	3.790836	3.976559	2.5718633	3.103504
3	1.849415	0.9332577	3.379418	3.581881	0.9698623	1.743047

Nevertheless, this approach faces the problem of not following any specific procedure for select and refine the minimal adequate model that represent the relationship of different covariates. To solve this problem Fraley et al. (2002, 2005, 2006, 2007) propose a set of techniques based on the selection of the most appropriate model and optimal number of clusters based on the Bayesian Information Criterion, computed in turn over several models and a range of values for different number of clusters (until we get the optimal results).

Figure 5.3 represents a scatterplot for each pair of covariates included in our analysis, allowing us to visually identify different clusters depending on the combination of values selected for each graph. Besides, we can appreciate clear linear relationships between variables like *nreporters* and *nreports*, both clearly conforming two different clusters. The package *mclust* implements in GNU R this analysis.

Mclust performs several runs, each one considering different shapes, relationships and number of components (covariates to be taken into account) to build each individual model. Finally, this method selects the model with the highest BIC, thus choosing the combination that better fits the

data set according to this criterion. Figure 5.4 shows all possible options, and as we can see, the tool selects the following option as the optimal solution in this case:

best model: ellipsoidal, equal shape with 2 components

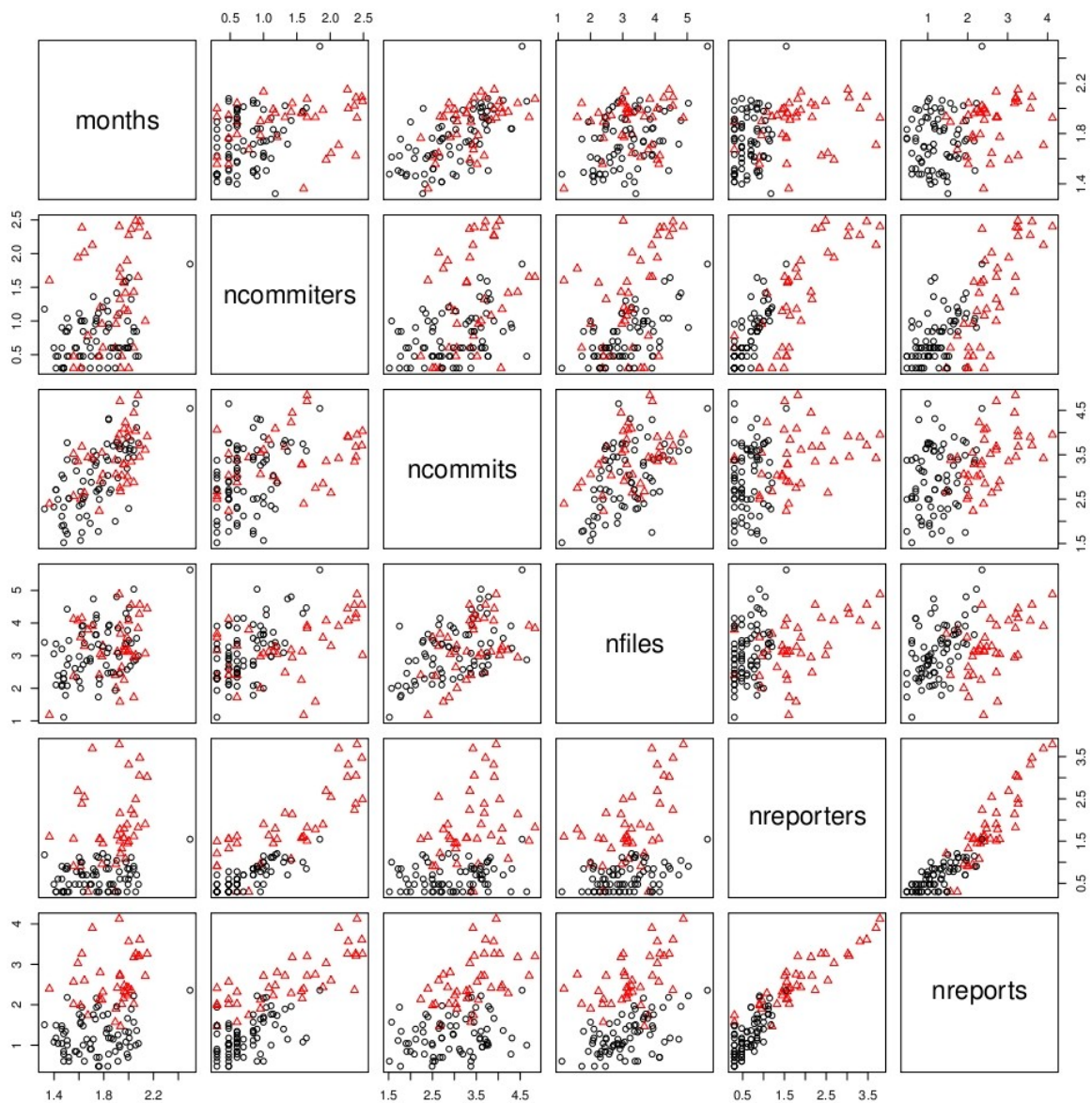


Figure 5.3 Scatterplot presenting relationships between pairs of covariates in our models. The clusters have been depicted according to the optimal solution found by the *mcluster* library.

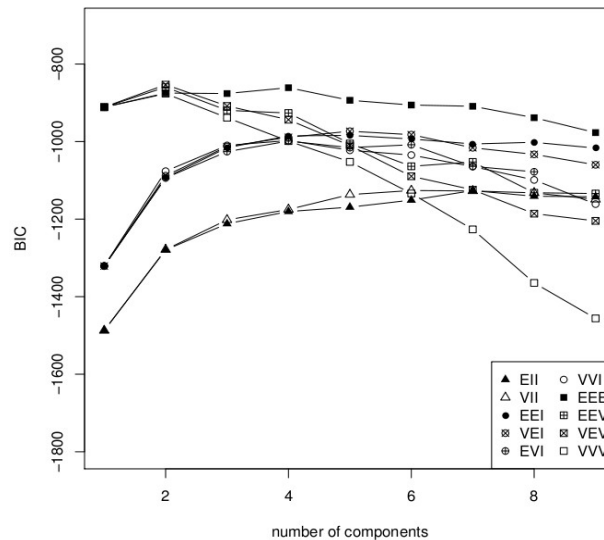


Figure 5.4 Graph depicting the complete set of combinations tested by *mcluster*. The highest BIC value corresponds to the *ellipsoidal, equal shape model with 2 components*.

Finally, Figure 5.5 presents a 3D plots depicting the suggested optimal solution for clustering our data sample, according to the clustering proposed by *mcluster*. A simple inspection of this graph will show that, though we can distinguish 2 different clusters according to the optimal model, the actual isolation level of each of these two groups is fairly low, suggesting that not all considered variables are playing a key role to conform the clusters, as we will see in the sequel.

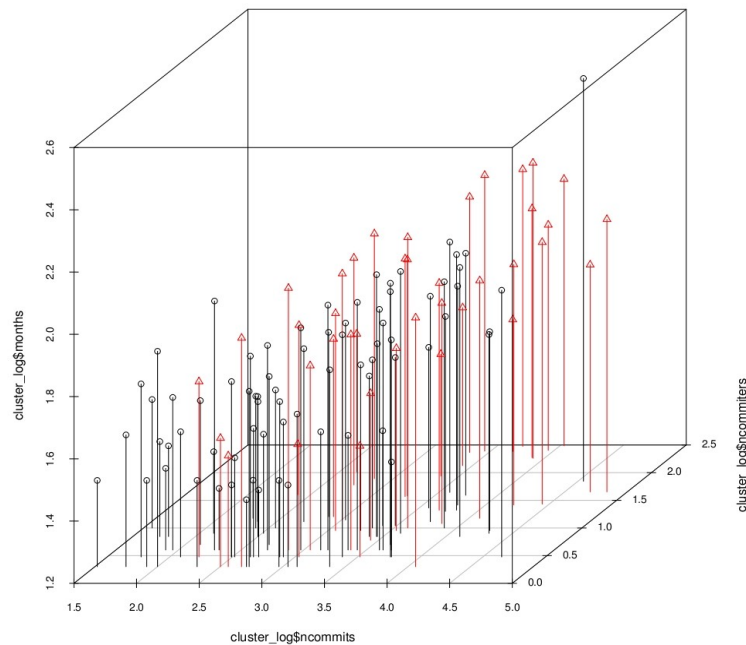


Figure 5.5 3D graph depicting the optimal solution found by *mcluster*

Everitt, B.S., Landau, S., and Leese, M. (2001), *Cluster Analysis*, London, UK: Arnold, 4th edition.

Gordon, A. (1999), *Classification*, Boca Raton, Florida, USA: Champan & Hall/CRC, 2nd edition.

C. Fraley and A. E. Raftery (2006). MCLUST Version 3 for R: Normal Mixture Modeling and Model-Based Clustering, Technical Report no. 504, Department of Statistics, University of Washington.

C. Fraley and A. E. Raftery (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American Statistical Association* 97:611:631.

C. Fraley and A. E. Raftery (2005). Bayesian regularization for normal mixture estimation and model-based clustering. Technical Report, Department of Statistics, University of Washington.

C. Fraley and A. E. Raftery (2007). Bayesian regularization for normal mixture estimation and model-based clustering. *Journal of Classification* 24:155-181.

5.3.3 Classification Trees

In the previous section, we found that optimal clustering for our data sample suggest that we have to simplify our explanatory model for the set of projects considered in this analysis. In order to discriminate which variables are playing a key role from the not significant covariates, we perform a classification or regression tree analysis on our data set.

Figure 5.6 presents the initial classification tree found when we consider all possible covariates.

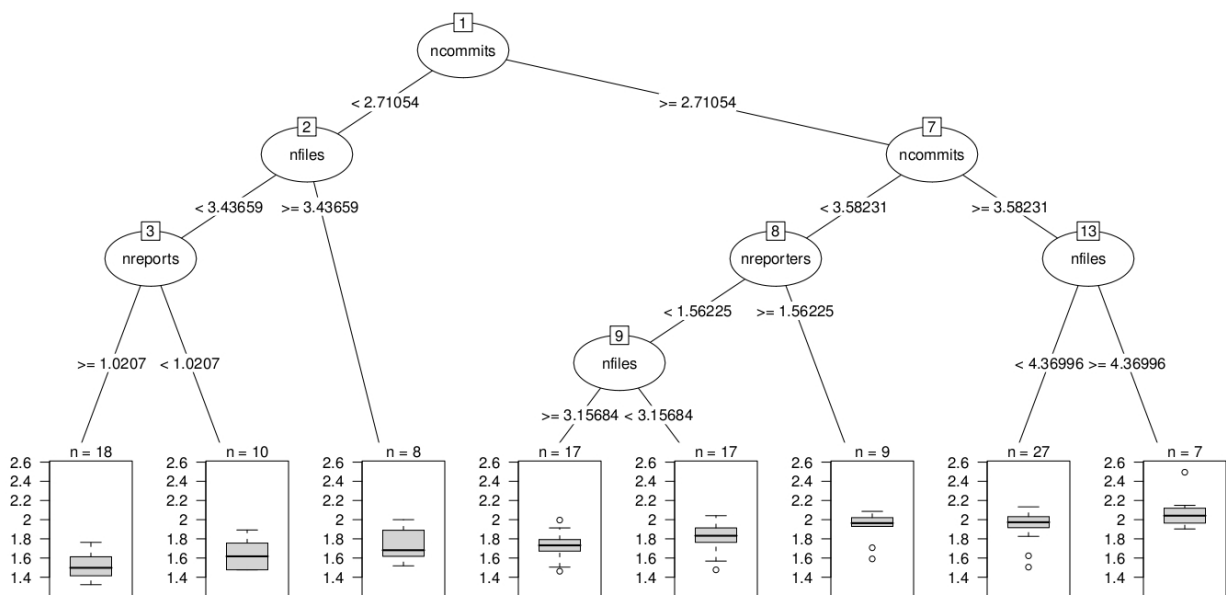



Figure 5.6 Complete classification tree for our data set. *Months* has been chosen as the dependent variable in this model, and the rest of covariates perform as descriptive parameters of the model.

The tool is able to make the best choices to select the nodes at which new branches should be taken, in order to classify our data set in different subpopulations. The final leaves show a box and whisker plot of the values of the dependent variable (*months* in this case) for each subgroup.

	<p style="text-align: center;">Classification Report</p> <p style="text-align: center;">Deliverable ID: D4.1</p>	Page : 42 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

Numerical values are the following:

node), split, n, deviance, yval
* denotes terminal node

- 1) root 113 5.3900820 1.788109
- 2) ncommits< 2.71054 36 0.9239939 1.598508
- 4) nfiles< 3.436588 28 0.5171723 1.558876
- 8) nreports>=1.020696 18 0.2656444 1.522682 *
- 9) nreports< 1.020696 10 0.1855027 1.624026 *
- 5) nfiles>=3.436588 8 0.2089150 1.737219 *
- 3) ncommits>=2.71054 77 2.5668890 1.876753
- 6) ncommits< 3.582308 43 1.1470640 1.798940
- 12) nreporters< 1.562252 34 0.7642643 1.766701
- 24) nfiles>=3.156843 17 0.2885053 1.720949 *
- 25) nfiles< 3.156843 17 0.4045878 1.812453 *
- 13) nreporters>=1.562252 9 0.2139599 1.920733 *
- 7) ncommits>=3.582308 34 0.8301895 1.975164
- 14) nfiles< 4.369961 27 0.4818108 1.946267 *
- 15) nfiles>=4.369961 7 0.2388702 2.086623 *

Now, we proceed to *prune* the complete tree, eliminating all covariates that do not play a significant explanatory role in the model following a step-wise simplification (automatically provided by GNU R functions) to compute the minimal adequate model in this case.

CP	nsplit	rel	error	xerror	xstd
1	0.35235078	0	1.0000000	1.0256908	0.12257329
2	0.10939268	1	0.6476492	0.7598182	0.10392952
3	0.03671679	2	0.5382565	0.6435117	0.08910186
4	0.03132412	3	0.5015397	0.6990892	0.09482811
5	0.02031665	4	0.4702156	0.7324534	0.10294485
6	0.01320411	5	0.4498990	0.7609752	0.10291888
7	0.01224940	6	0.4366949	0.7528933	0.10153084
8	0.01000000	7	0.4244455	0.7566068	0.10138998

Thus, we can see that the minimal error (both in absolute and normalized values) corresponds to a tree with only 3 splits (or nodes). Figure 5.7 shows the graphical representation of this optimal result.

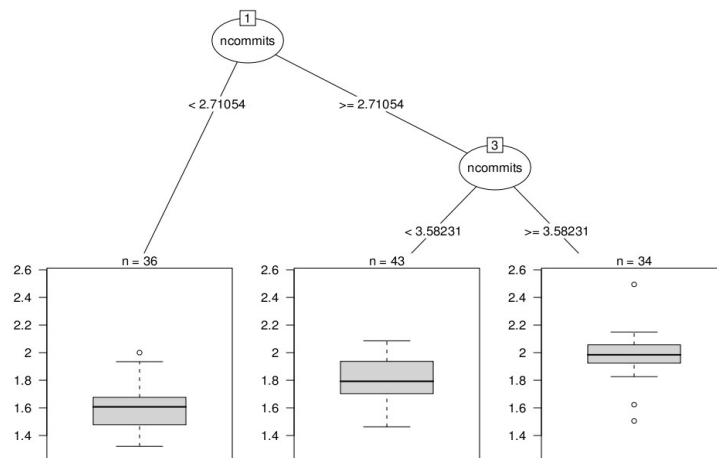



Figure 5.7 Optimal solution found for the classification tree in our model

In conclusion, from the results obtained in the statistical techniques applied to cluster and classify our projects, we can infer that the only significant covariate is the number of commits, so that we can predict the longevity of the project according to the conditional values expressed in the optimal classification tree.

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 44 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

6. APPENDIXES

6.1 SQL STATEMENTS FOR COMPLEX VARIABLES

This section describes SQL statements used to obtain the different values of the complex variables based on the FLOSSMetrics database design. Please note that FLOSSMetrics database is managed by the MySQL database management system (DBMS) and these queries are specific of this system. Therefore, some of this queries might not work properly in other DBMSs.

6.1.1 Evolution of first reports submitted by registered users


- BTS Database -

```
SELECT g.yearmonth,count(g.SubmittedBy)
FROM (
  SELECT date_format(MIN(b.DateSubmitted), '%Y%M') yearmonth, b.SubmittedBy
  FROM Bugs b, GeneralInfo gi
  WHERE gi.Tracker='Bugs'
  GROUP BY b.SubmittedBy
) g
GROUP BY g.yearmonth
ORDER BY g.yearmonth;
```

6.1.2 Evolution of fist commits submitted by registered users

- SCM Database -

```
SELECT g.yearmonth, count(g.committer_id)
FROM (
  SELECT s.committer_id, date_format(min(s.date), '%Y%M') yearmonth
  FROM scmlog s, actions a, file_types ft
  WHERE s.id=a.commit_id AND a.commit_id=ft.commit_id AND a.file_id=ft.file_id
  AND ft.type='code'
  GROUP BY s.committer_id
) g
GROUP BY g.yearmonth
ORDER BY g.yearmonth;
```

	<p style="text-align: center;">Classification Report</p> <p style="text-align: center;">Deliverable ID: D4.1</p>	Page : 45 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

6.1.3 Evolution of new core members

- SCM Database -

This query stores the values in a temporary table named 'tbcore'

```

SELECT tbaux.myyear,tbaux.committer,tbaux.acusum,tbaux.mycom commits
FROM
(SELECT g.myyear,IF(g.committer_id IS NULL,'-',g.committer_id) committer,
IF(g.committer_id IS NULL,@sumacu:=0,@sumacu:=@sumacu+g.mycom) acusum,
IF(g.committer_id IS NULL,0,g.mycom) mycom
FROM
(SELECT @sumacu:=0) r,
(SELECT date_format(s.date, '%Y') myyear, s.committer_id, COUNT(s.id) mycom
FROM scmlog s
GROUP BY date_format(s.date, '%Y'), s.committer_id WITH ROLLUP) g
ORDER BY g.myyear, g.mycom) tbaux,
(SELECT date_format(s.date, '%Y') myyear, count(s.id) , COUNT(s.id)*20/100
nocore
FROM scmlog s
GROUP BY date_format(s.date, '%Y')
ORDER BY myyear
) tbttotal
WHERE tbaux.myyear=tbttotal.myyear and tbaux.acusum>tbttotal.nocore ;

```

```

SELECT Q4.coreyear, count(Q4.committer)
FROM (
SELECT MIN(tbcore.myyear) coreyear, tbcore.committer tbcore
FROM tbcore
GROUP BY tbcore.committer
ORDER BY coreyear
) Q4
GROUP BY Q4.coreyear;

```


6.1.4 Evolution of core members leaving the core team

- SCM Database -

```

SELECT Q5.coreyear, COUNT(Q5.committer)
FROM (
SELECT y1.myyear+1 coreyear, y1.committer
FROM
(SELECT concat( tbcore.committer,tbcore.myyear) key1,tbcore.myyear,
tbcore.committer
FROM tbcore) y1 LEFT JOIN
(SELECT concat( tbcore.committer,tbcore.myyear) key2,tbcore.myyear,
tbcore.committer
FROM tbcore)y2 on y1.key1+1=y2.key2
WHERE key2 IS NULL AND y1.myyear NOT IN (SELECT MAX(myyear) FROM tbcore)
)Q5 GROUP BY Q5.coreyear;

```

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 46 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public


6.1.5 Evolution of the balance in the core team

- SCM Database -

```

SELECT tbq4.coreyear,
  IF (tbq5.corevalue IS NULL, tbq4.corevalue, tbq4.corevalue-tbq5.corevalue)
diffnewstopped FROM
(SELECT Q4.coreyear, COUNT(Q4.committer) corevalue
FROM (
  SELECT MIN(tbcore.myear) coreyear, tbcore.committer
FROM tbcore
GROUP BY tbcore.committer
ORDER BY coreyear
) Q4 GROUP BY Q4.coreyear)tbq4
LEFT JOIN (
  SELECT Q5.coreyear, COUNT(Q5.committer) corevalue
FROM (
  SELECT y1.myear+1 coreyear, y1.committer
FROM (
  SELECT CONCAT( tbcore.committer,tbcore.myear) key1,tbcore.myear,
tbcore.committer
FROM tbcore) y1
LEFT JOIN (
  SELECT concat( tbcore.committer,tbcore.myear) key2,tbcore.myear,
tbcore.committer
FROM tbcore)y2
ON y1.key1+1=y2.key2
WHERE key2 IS NULL AND y1.myear NOT IN (SELECT MAX(myear) FROM tbcore)
) Q5 GROUP BY Q5.coreyear) tbq5
ON tbq4.coreyear=tbq5.coreyear
UNION
SELECT tbq4.coreyear, tbq5.corevalue*-1 diffnewstopped
FROM (
  SELECT Q4.coreyear, COUNT(Q4.committer) corevalue
FROM (
  SELECT MIN(tbcore.myear) coreyear, tbcore.committer
FROM tbcore
GROUP BY tbcore.committer
ORDER BY coreyear
) Q4 GROUP BY Q4.coreyear)tbq4 RIGHT JOIN
(SELECT Q5.coreyear, COUNT(Q5.committer) corevalue
FROM (
  SELECT y1.myear+1 coreyear, y1.committer
FROM (
  SELECT concat( tbcore.committer,tbcore.myear) key1,tbcore.myear,
tbcore.committer
FROM tbcore) y1 LEFT JOIN
(SELECT concat( tbcore.committer,tbcore.myear) key2,tbcore.myear,
tbcore.committer
FROM tbcore)y2 on y1.key1+1=y2.key2
WHERE key2 IS NULL AND y1.myear NOT IN (SELECT MAX(myear) FROM tbcore)

```

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 47 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

```
)Q5 GROUP BY Q5.coreyear) tbq5
ON tbq4.coreyear=tbq5.coreyear
WHERE tbq4.coreyear IS NULL;
```

6.1.6 Average of committers longevity

- SCM Database -

```
SELECT SUM(total.sum_mounths)/COUNT(total.list_committers)
FROM (
  SELECT new.committer_id list_committers, COUNT(new.committer_id) sum_mounths
  FROM (
    SELECT committer_id, date
    FROM scmlog
    GROUP BY committer_id, year(date), month(date)
  ) new
  GROUP BY new.committer_id
) total;
```

6.1.7 Evolution of code contributors who submitted patches and changes

- BTS Database -

```
SELECT date_format(b.DateSubmitted, '%Y') myyear, COUNT(b.SubmittedBy)
FROM Bugs b, GeneralInfo g
WHERE g.Tracker='Patches'
GROUP BY date_format(b.DateSubmitted, '%Y');
```


6.1.8 Total code contributors who submitted patches and changes

- SCM Database -

```
SELECT 'CVSAnaly', date_format(s.date, '%Y') myyear,
COUNT(DISTINCT s.committer_id)
FROM scmlog s, actions a, file_types ft
WHERE s.id=a.commit_id AND a.commit_id=ft.commit_id AND a.file_id=ft.file_id
AND ft.type='code'
GROUP BY date_format(s.date, '%Y');
```

- BTS Database -

```
SELECT 'BTS', date_format(b.DateSubmitted, '%Y') myyear,
COUNT(DISTINCT b.SubmittedBy)
FROM Bugs b, GeneralInfo g
WHERE g.Tracker='Patches'
GROUP BY date_format(b.DateSubmitted, '%Y');
```

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 48 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

6.1.9 Evolution in number of events

- SCM Database -

```
SELECT 'CVSAnaly',date_format(s.date, '%Y') myyear,
COUNT(s.id)/COUNT(DISTINCT s.committer_id) avg_commitxcommitter
FROM scmlog s
GROUP BY date_format(s.date, '%Y');"
```

- MLS Database -

```
SELECT 'MLS', date_format(m.first_date, '%Y') year,
COUNT(DISTINCT m.message_ID)/COUNT(DISTINCT mp.people_ID)
FROM messages m, messages_people mp
WHERE m.message_ID=mp.message_ID
GROUP BY date_format(m.first_date, '%Y');
```

- BTS Database -

```
SELECT 'BTS', date_format(b.DateSubmitted, '%Y') myyear,
COUNT(b.idBug)/COUNT(DISTINCT b.SubmittedBy)
FROM bicho.Bugs b
GROUP BY date_format(b.DateSubmitted, '%Y');
```

6.1.10 Evolution in number of commits


- SCM Database -

```
SELECT date_format(s.date, '%Y') myyear,
COUNT(s.id)/COUNT(DISTINCT s.committer_id) avg_commitxcommitter
FROM scmlog s, actions a, file_types ft
WHERE s.id=a.commit_id
AND a.commit_id=ft.commit_id AND a.file_id=ft.file_id AND ft.type='code'
GROUP BY date_format(s.date, '%Y');
```

6.1.11 Percentage of people working in old releases

- BTS Database -

```
SELECT COUNT(DISTINCT b.SubmittedBy)
FROM Bugs b,Changes c
WHERE b.Status='CLOSED' AND c.OldValue <> 'CLOSED'
AND c.Field='status_id' AND b.idBug=c.idBug;
```

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 49 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

6.1.12 Territoriality

- SCM Database -

```
SELECT (COUNT(g.file_id) / (SELECT COUNT(DISTINCT file_id) FROM actions)) * 100
FROM (
  SELECT a.file_id, COUNT(DISTINCT s.committer_id)
  FROM actions a, scmlog s
  WHERE a.commit_id=s.id
  GROUP BY a.file_id
  HAVING COUNT (DISTINCT s.committer_id)=1
) g;
```

6.1.13 Activity per committer

- SCM Database -

```
numcommitter=
SELECT COUNT(DISTINCT s.committer_id) numcommitteract
FROM scmlog s
WHERE date_format(date, '%Y%m%d') >= (
  SELECT
    CONCAT(date_format(MAX(s.date), '%Y')-1,
           date_format(MAX(s.date), '%m'),
           date_format(MAX(s.date), '%d'))
  FROM scmlog s);

SELECT (SUM(m.sloc)/$numcommitter) linesxcommitter
FROM metrics m;
```

- MLS Database -

```
SELECT (COUNT(m.message_id)/$numcommitter) messagesxcommitter
FROM messages m;
```


- BTS Database -

```
SELECT (COUNT(b.idbug)/$numcommitter) bugsxcommitter
FROM Bugs b;
```

6.1.14 Number of lines per committer

- SCM Database -

```
SELECT SUM(m.sloc) / (
  SELECT COUNT(DISTINCT s.committer_id) numcommitteract
  FROM scmlog s
```

	<p>Classification Report</p> <p>Deliverable ID: D4.1</p>	Page : 50 of 50
		Version: 2.0
		Date: Oct 31, 09
		Status : Final Confid : Public

```

WHERE date_format(date,'%Y%m%d')>=
(SELECT
  CONCAT(date_format(max(s.date),'%Y')-1,
    date_format(max(s.date),'%m'),
    date_format(max(s.date),'%d'))
  FROM scmlog s) linesxcommitter
FROM metrics m;

```

6.1.15 Evolution of first reports submitted by registered users

```

- BTS Database -

SELECT
  COUNT(DISTINCT a.file_id)/
  (SELECT COUNT(DISTINCT file_id)
   FROM actions)
FROM actions a, (
  SELECT s.id, s.committer_id numcommitteract
  FROM scmlog s
  WHERE date_format(date,'%Y%m%d')>= (
    SELECT
      CONCAT(date_format(max(s.date),'%Y')-1,
        date_format(max(s.date),'%m'),
        date_format(max(s.date),'%d'))
    FROM scmlog s)
) g, file_types ft
WHERE a.commit_id=g.id AND a.file_id=ft.file_id AND ft.type='code';

```