Executable source code and non-executable source code: analysis and relationships

Gregorio Robles, Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos
Grupo de Sistemas y Comunicaciones
Tulipan s/n, 28933 Mostoles (Madrid), Spain
{grex,jgb}@gsyc.escet.urjc.es

Abstract

The concept of source code, understood as the source components used to obtain a binary, ready to execute version of a program, comprises currently more than source code written in a programming language. Specially when we move apart from systems-programming and enter the realm of end-user applications, we find source files with documentation, interface specifications, internationalization and localization modules, multimedia files, etc. All of them are source code in the sense that the developer works directly with them, and the application is built automatically using them as input.

This paper discusses the relationship between 'classical' source code (usually written in a programming language) and these other files by analyzing a publicly-available software versioning repository. Aspects that have been studied include the nature of the software repository, the different mixtures of source code found in several software projects stored in it, the specialization of developers to the different tasks, etc.

1. Introduction

Software development has evolved from command-line applications to huge end-user applications full of graphics and multimedia elements. Following this trend, software development has changed from a task done mainly by software developers to an activity requiring the coordinated work of many different groups that are commited to various tasks: internationalization and localization (i18n and l12n), graphic design, user interface design, technical and end-user documentation writing, creation of multimedia elements, and many others.

'Classical' source code analysis has focused on the output of the work done mainly by software developers, source code written in a programming language. The other elements mentioned above, built usually by professionals with other backgrounds, have usually not been considered, even though they are in many cases a fundamental part of the application. We consider that all those elements are also an integral part of the software development process, and propose the beginning of a path for its integral study, by looking at the interdependences existing between all of them. In this sense, our intention is to extend, for the purposes of the analysis of the software production process, the concept of source code to all those other elements different from pure programming code.

There is plenty of literature devoted to the analysis of source code, both in the commercial software environment and in the libre software[1] world, but we've found few articles focused on the aspects that we are considering in this paper.

Some previous studies have attented to gain some knowledge on the persons who are working on a self-organized project. One attempt is the one given by Capiluppi et al.[2] which analyzes changelogs in a almost-automatic way. But this analysis is very limited and unaccurate in its scope and does not offer a possibility to include a deeper analysis of the sources as we intend in this paper.

A better source of data for our interests are other publicly-available repositories[9], specially the versioning systems used for software development. These type of systems allow to monitor the whole development process, including file types and the developers. Another argument for their use is that evolutionary studies are easy to perform as all the previous states are avail-

[1] Throughout this article we use the term 'libre software' to refer both to free and open source software.

able. Godfrey et al.[3] present such an evolutionary study for the Linux kernel, although their primary aim was centered on performing a "pure" source code and dependency analysis, so this work could be classified into the field of "classical" software evolution theory[5]. Our goal differs from this study in two main points. First, we are more interested in end-user applications rather than low-level programs as the Linux kernel is as these have a bigger amount of source elements different from "pure" source code. Second, we also want to research the human interactions and evolution in such a software project. In this sense, there exists already some literature that has performed some analysis on self-organizing development groups using social network analysis techniques[6] and proposing ways of identifying the community structure[4].

In our study, we discriminate several file types, corresponding different 'kinds' of source code. From the analysis of such files and their evolution over time we may infer the importance that a given software program (or project) gives to different activities like documentation, translation, user interface design or multimedia development.

This paper proposes a methodology for the study, and a software that makes it possible when the sources are stored in a CVS repository and certain common (at least in libre software projects) conventions are used.

In the following section the methodology of this paper will be presented. Next, a case of study will be shown: the KDE project, a libre software desktop environment with hundreds of applications and a large development community around it. Results on the modules and on the developers of the KDE CVS repository will be given next. Finally, some conclusions and further work will be discussed. This paper includes also an appendix containing some additional information about the methodology.

2. Methodology

The methodology described in this paper is based on the analysis of CVS[2] log entries. Retrieving data from a versioning system makes it possible not only to have the latest version of the source code, but also to have the possibility of fetching data in any point in time since the repository was set up. Hence, an evolutionary study is feasible.

The methodology is implemented in an automated way by the CVSAnalY tool [8]. In CVSAnalY any interaction (also called commit) performed by a commiter[3] in the CVS repository is logged with the following data associated: commiter name, date, file, revision number, lines added, lines removed, and an explanatory comment introduced by the commiter. It is of great importance for the goal of this paper to note that commiters should not be taken only as software developers in the sense of programming code generators; commiters can also be persons devoted to other tasks such as translation or graphical design. Since it is possible to know, from the logs, which commiters have done which actions, it is also possible to correlate those files with them and classify them (the commiters) according to the task they are fulfilling.

There is also some file-specific information in the CVS logs that can be skimmed, like for instance whether the file has been removed[4]. An analysis of the file name makes it possible to sort files by type, so that programming-language files can be separated from translation files and so on. This has been made using simple heuristics that pay attention primarily on the file extension, or search for other common patterns that make identification possible. In the next subsection file type identification will be discussed further.

If needed, the human-inserted comment can also be parsed in order to learn whether the commit corresponds to an external contribution, or even to an automated script. This comment is usually forwarded to a mailing list so that developers keep track of the latest changes in CVS. Some projects have also conventions so that certain commits do not produce a message to the mailing list as it is supposed that the action they have performed does not require any notification. 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 commiters may also use them. In a large community -as it is the case for the ones we are researching- we can argue that "silent" commits can be considered as not contributory. Therefore, we have set a flag for such commits in order to compute them separately or leave them out completely.

Once the CVS logs have been parsed, and a database has been fed with it, a postprocess stage is performed. Several scripts querying the database perform statistic analysis, calculate several inequality and concentration indices, and generate graphs for the evolution in time for a couple of interesting parameters (commits, commiters, LOCs...). Results are shown through a publicly

[2]The Concurrent Versions System (CVS) is the most popular versioning repository used in the libre software world.

[3]A commiter is a person who has write access to the repository and does a commit on it at a given time.

[4]In CVS there is actually no file removal. Files that are not required anymore are stored in the Attic and could be called back anytime in the future.

accessible web interface that permits an easy inspection of the whole repository (general results), a single module or by commiters. Therefore these results themselves are again available for remote analysis and interpretation by project participants and other interesting parties.

3. Case of study: KDE

KDE is a libre software project with the goal of building a libre software graphical desktop environment for UNIX-like operating systems. The desktop and its applications (as for instance their own office suite called KOffice) are built by making use of their application development framework. A big community has flourished in the last years around KDE: the number of commiters almost reaches 1,000 persons.

A CVS repository is internally organized in modules. Modules may contain several applications usually of the same application family, so there is a KOffice module which groups the office suite applications (word processor, spreadsheet, presentation program, etc.). Other modules serve for the project's own administrative means and there exists a module that is used to store all the translation files.

Table 1. General statistics for the KDE project

Number of modules	79
Number of commiters	915
Number of commits	2,935,436
Number of files	175,657
Lines added	106,036,517
Lines removed	73,534,466
First commit	1997-04-09
Last commit considered	2004-03-22
Number of days	2,539

Figure 1 presents a wheighted distribution of the file types stored in the KDE CVS repository. The wheight that has been used is the number of commits done to files that correspond to each file type. This gives us an idea of the activity that is done around any given file type that we are investigating.

A first impression offers us already some information. First, KDE is clearly a software development project (red is the biggest portion), but the effort invested into software development does not reach by far 50%. Second, the amount of translations is a good indicator of the wide spread of the KDE project around the globe. Third, documentation and images are also heavily represented. Fourth, we are able to see that the amount of sound (multimedia) files is minimal, an evidence that KDE is not a multimedia project, while the user interface fraction (around 15%) is big enough to properly argument that it is actually a desktop-targeted environment. Finally, the fraction corresponding to the unknown section lies under 3%.

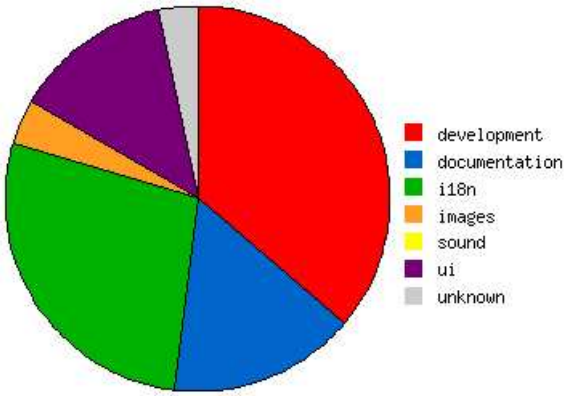

Figure 1. Commits by file type for the KDE repository.

Figure 2 presents the distribution of commits per module for the selected file types, being both axes in logarithmic scale. We expected to find a powerlaw distribution as it is commont in other distributions like computer networks[1]. But the graphs point out a Poisson distribution with the interesting case of the purple line (the one that corresponds to the i18n files) which has two very differentiated regions, similar to the Poisson distribution found for instance in the number of synonyms a word has in the English language.

4. Modules

Our methodology offers the possibility of studying each module and commiter for its own. This will make possible to classify modules and commiters depending on its composition in the first case and on the tasks it is devoted to in the second one. This way the most active file types give us an idea of the nature of modules and commiters and further research may also allow to know their specialization.

As an example of the analysis that can be done on modules we will pay attention on the KOffice module. KOffice is the office suite developed by the KDE project. Figure 3 shows the distribution of the different file types in a pie for KOffice, from which we

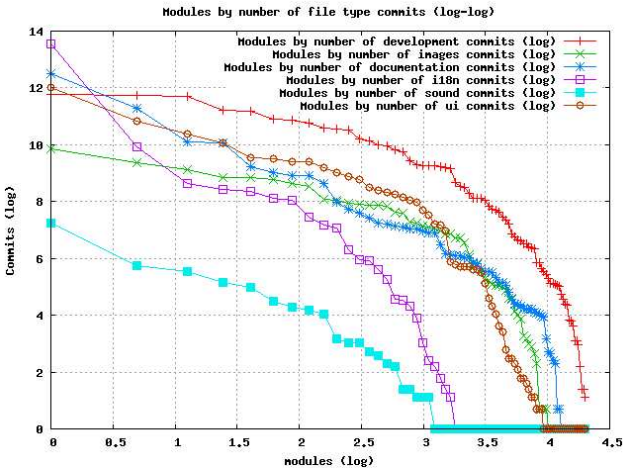

Figure 2. Log-log representation of file types among KDE modules.

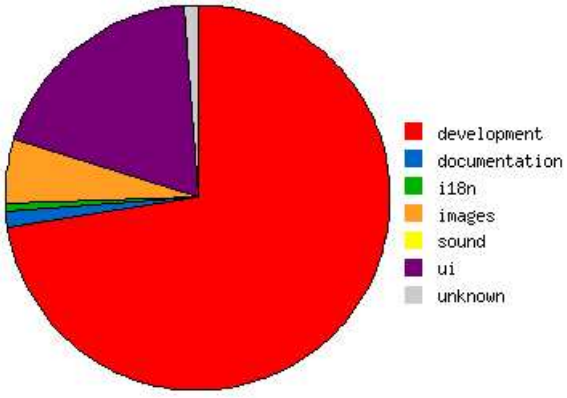

Figure 3. Commits by file types for KOffice.

may infer that it is primarily a development project (red), although the user interface (purple) portion is not negigible. All other elements considered in this study (images, documentation, translations and sound) are very rare. The fact that commits done to translation files are so seldom is due to the existence of an external module in the KDE project which centralizes all the translations. Later these translations are automatically joined with the sources. Commits done to unidentified file types ('unknown') correspond also to a minimal portion of the pie.

The distribution depends of course on the module that we are studying. For instance, the module that contains all the translations done project-wide and hence has a predominant green portion in its pie. Usually modules that contain applications or a set of applications have a pie that looks similar to the one shown for KOffice (mostly red), while modules dedicated to web pages and documentation have a clear blue (documentation) and orange (images) flavour. Interesting are some minor modules like kdeedu (a KDE subproject that contains applications suited for education) which has an important portion of multimedia elements (that appear yellow and are labeled as sound in the pie).

Besides pies another way of data visualization is given by heat maps. Heat maps can give an idea of the specialization of commiters that work on a given module. The idea is to show visually the correlation given by a percentage that exists between file types. If this percentage is high it will correspond in the heat map with a hot zone that is modelled as a clear color (yellow or orange), while small values will correspond to cold zones given by obscure colors (like blue or grey). Black has been reserved as background color and also appears when no files of a give file type exist or in the diagonal (that really should be yellow as all its values are of 100%) in order to make heat map reading easier.

Having a look at Figure 4 will make it clearer. The first row shows the correlation of documentation with all other file types considered in this study. As noted before, the intersection of documentation with itself has been left black although it should really be yellow as the correlation with itself is 100%. The second column gives the percentage of the commits to documentation that commiters have done to documentation and that have contributed both to documentation and images in the KOffice module. As it can be observed this zone is very cold (blue is given when values lie between 0 and 20%). The next column gives us information about the number of commits done by those commiting files to documentation and translations. As translation files are not included in the KOffice module, the zone is very cold as it is given by the grey color. On the other hand the common contribution of commiters to documentation that have also contributed to user interface and to development (fourth and sixth column of the first row) is very high (more than 80%) in the case of documentation. As there are no sound files in that time period, the whole sound column as well as its row have been left black. Interesting is also the case for the 'unknown' type as it may infere also information as where to locate these type of files that are not well sorted by our heuristics. In the case of documentation we can see that this value is rather low (cold).

Note that the relation given in the heat maps has not to be symmetric, because although the number of

commits commiters have done to documentation and translation is the same, the total number of commits to documentation and to translation differ.

As we have at our disposal data from the project history, we may also study the evolution of the specialization. Therefore we have taken 10 time-equally large time slots from the first commit of a module to its present state and generated a heat map for each time slot. In the case of KOffice, the first commit was done 1998-04-18 while the last one considered in this study dates from 2004-03-22. Hence, each time slot's interval supposes around 216 days (little more than 7 months) of activity.

Figure 4 corresponds to the time period from 2000-08-30 to 2001-04-04 while figure 5 corresponds to the time period three yearse later (exactly from 2003-08-18 to 2004-03-22). A close look at both maps throws that there is a slight specialization (hoter zones are more rare in the newer one) as well as a project expansion (there are more file types in the latter than in the former one).

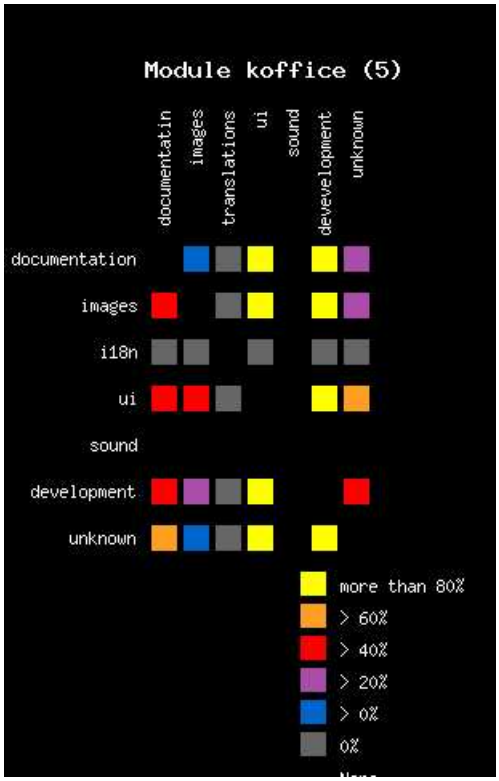

Figure 4. File type correlation heat map for the 5th time slot.

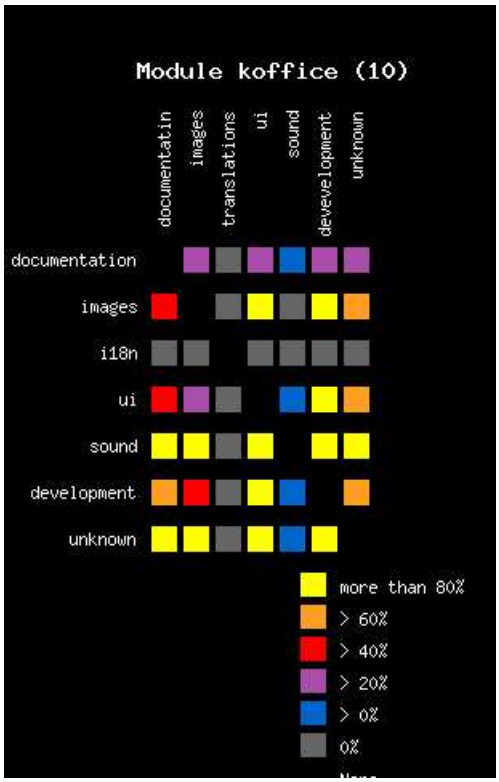

Figure 5. File type correlation heat map for the 10th (last) time slot.

5. Commiters

Write access to the versioning system is not given to anyone. Usually this privilege is only given to contributors who reach a compromise with the project and the project's goals. 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 (commiters) are always welcome. It is a widely accepted practice to mark an external contribution when commiting it with an authorship attribution, so we have constructed certain heuristics to find and mark commits due to such contributions, although in this study we haven't filtered those contributions out.

In the following scatterplots any point corresponds to a commiter. The color of the point is given by the file type where the commiter is being more active. Color assignation follows the rules used in the pies shown in the previous sections and are summarized in table 2. In one axis of the scatterplots the contributions to a given file type are shown, while in the other axis contributions to another file type are given. The distribution

of commiters in the XY space may give us an idea of the specialization of commiters as well as the possible relationships that may exist between different file types.

In order to make the data offered by the scatterplots easier to work with we have taken the natural logarithm of the commits done by commiters to the whole repository. This means that only active commiters will be shown (those who have at least one commit in any of the two categories considered) and that the axis contain those commiters who have commited only one commit to a file type and one or more commits to the other one. This confronted us with the problem of commiters who haven't done commits to one of the file types considered but with a considerable amount to the other. In order to show these commiters too we have considered commiters that have twenty or more commits [5] in one file type to have at least one commit to the other (if no commit had been done, this was added automatically). This should not be a dramatic distorsion of the data and would give us valuable information specially about specialization of commiters.

Table 2. Colors used to identify file type where most active in the scatterplots

Color	File type
Red	Development
Blue	Documentation
Green	Translations
Orange	Images
Yellow	Multimedia
Purple	User interface
Gray	Unknown

The first scatterplot we are going to considered is shown in Figure and presents development commits in the X axis and documentation in the Y axis. There are several interesting facts that can be extracted from this figure. First, that the development 'population' (red points) is by far bigger than any other one. Second, that there is a natural division between documentators (blue) and developers (red) given by our way of coloring commiters by its highest contributing file type. Interesting on the other hand is the location of commiters whose primary task is none of these two: translators (green) are generally grouped with documentators, while those who work on the user interface (purple) appear in the red development dust. Third, that among the most contributing commiters (log(commits) ¿ 10, also more than 20,000 commits) to the development file type we can find nine persons, but only five of them having development as their first activity. Two more are mainly translators (green points) and the other two are primarily documentators (blue points).

[5] ln(20) is almost 3.

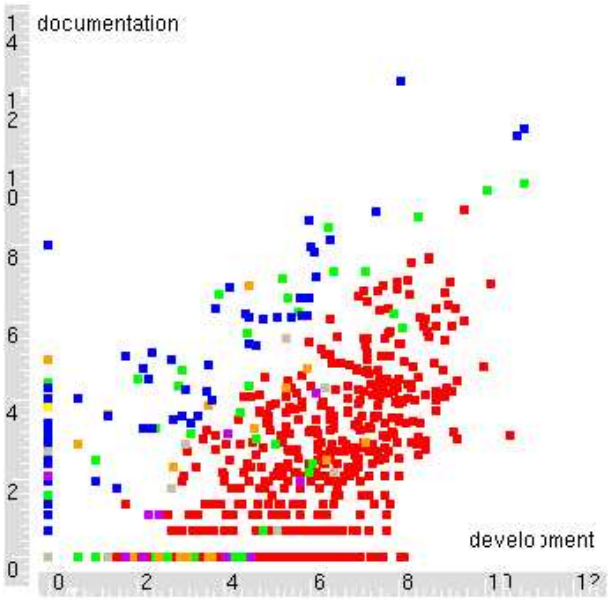

Figure 6. Documentation vs development scatterplot.

Figure maintains the development commits in the X axis and sets in the Y axis the ones related to translations. We see another time that there are several patterns that points with same color follow. Interesting is the fact that the Y axis contains only green points, meaning that many commiters do only translations tasks. This can be considered as a way of interfering some specialization in a project: almost half of the translators in the KDE project do not do any development activity. The situation of the blue points (documentation) in this scatterplot is also of great interest: there is a first group that lies in between translators (green points) and developers (red points). The interpretation for this is not straightforward. One possibility is that there is a natural tendency to get integrated into the project that starts by doing some translation work (which is pretty simple as it only requires to have some knowledge of the English and the own langauge), then by making some supporting task as documentation (which includes web pages) and finally landing developing software (which requires to have some knowledge on the platform and the technologies that are used, as well as some not-so-easy-to-acquire skills).

Another curious fact about this scatterplot is that there are almost any big contribution from commiters which are not green, blue or red.

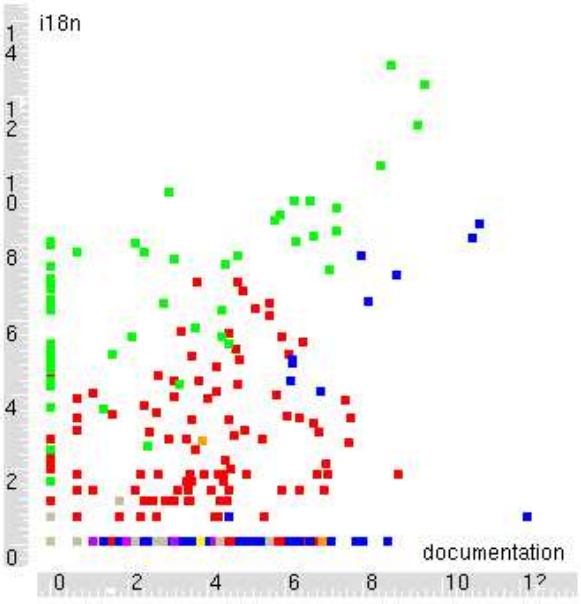

Figure 7. Translation vs documentation scatterplot.

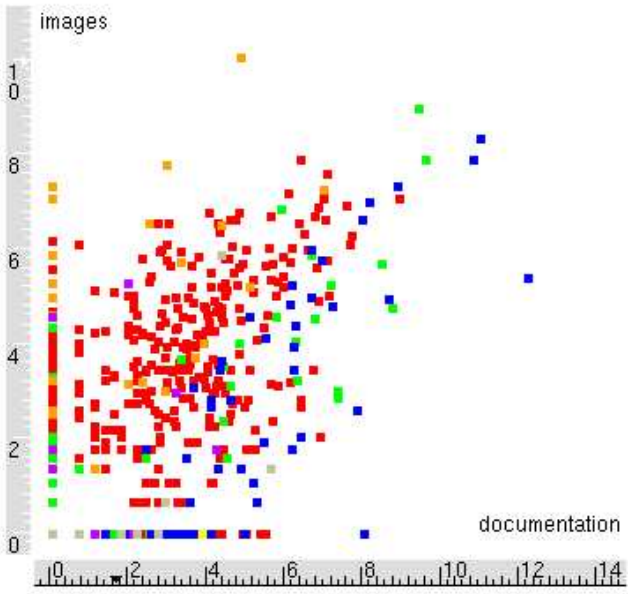

Figure 8. Images vs documentation scatterplot.

As aforementioned one of the backdraws of our identification heuristics is that it is hard to divide documentation and images clearly as some images correspond to documentation. This is the case for instance for web pages or technical documentation in XML. Figure may throw some light into this problem. The first aspect that is worth mentioning is that documentators rarely do big contributions in documentation and not inserting images. All major documentation contributors have also an appreciable number of images commits. The second one is that many developers contribute both documentation and images -really more images than documentation as the points are shifted more towards the Y axis. As observed in a previous scatterplot, translators are also more tied to documentators than to images.

The last scatterplot in Figure highlights the commiters that work on user interface tasks and development. The facts that can be seen here is that the number of persons dedicated to design and implement the user interface is rather small. Interesting is the fact that there is a noteworthy tendency that shows that while developers do have a bigger contribution to development their contribution to user interface file types also increases. This may be interpreted in the sense that besides the very specialized group that works on ui, all others start first by developing in the classical sense and as time passes and they acquire experience they also work on user interface.

It is also possible to obtain a scatterplot of commiters for a module and see if it corresponds to the general tendency. Among the interesting facts that we may look at is to see if there are differences between a local color choice (in the sense that only commits by a commiter to the module are taken into account) and a global color choice (where all commits made by a commiter to the repository are considered). The number of "color changes" and the file types that are most affected may allow us to infere some conclusions.

6. Conclusions and further work

Doing an examination of source code in the extended sense allows us to discriminate several type of files that reveal several "types" of source code. A deep study of them and of their evolution may help us infering what importance a program (or a project) attributes to certain tasks beyond generating source code in the sense of writing in a programming language. These activities include documentation, translation, user interface design, generation of multimedia elements, etc. This paper also proposes a methodology and a software that

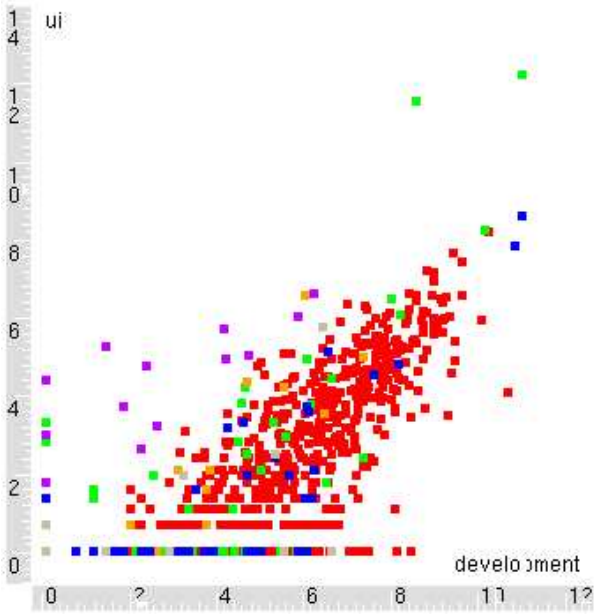

Figure 9. User interface vs development scatterplot.

implemts it in the case that the sources are stored in a CVS versioning system and that common conventions in the libre software world are used.

One of the main goals of this paper and which should be further researched is the possibility to use objective criteria to characterize projects based on its activity in the aforementioned areas and study the evolution of these activities over time. If data is available we can proceed to make the same study for part of the project (modules or subprojects) and even for the persons that are working on them (in the case of a CVS system these are commiters). Our first attempt has been to characterize them by assigning colors to the different tasks that we are considering and being able to visually recognize what type of module/commiter we have. Pending work includes studying correlations between modules and commiters.

Future research should also focus on the evolution of modules and commiters in time, although some aspects have been pointed out in this paper specially in the case of commiters. The scatterplots have shown that there are several tendencies among the behaviour of commiters although these have not been proved in a deterministic way. Hence, we argue that many commiters evolve from translators to documentators and finally to development tasks. This same behaviour arises with user interface, that require some previous activity in the development area.

References

[1] R. Albert, A. L. Barabsi, H. Jeong, and G. Bianconi. Power-law distribution of the world wide web. Science, 287, 2000.

[2] A. Capiluppi, P. Lago, and M. Morisio. Evidences in the evolution of os projects through changelog analyses. 2003.

[3] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. 2000.

[4] J. M. Gonzlez-Barahona, L. Lpez-Fernndez, and G. Robles. Community structure of modules in the apache project. 2004.

[5] M. Lehman, J. Ramil, P. Wernick, and D. Perry. Metrics and laws of software evolution - the nineties view. 1997.

[6] L. Lopez, J. M. Gonzalez-Barahona, and G. Robles. Applying social network analysis to the information in cvs repositories. In Proceedings of the International Workshop on Mining Software Repositories, 26th International Conference on Software Engineering, Edinburg, Scotland, UK, 2004.

[7] G. Robles, J. M. Gonzalez-Barahona, and R. A. Ghosh. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. In Proceedings of the International Workshop on Mining Software Repositories, 26th International Conference on Software Engineering, Edinburg, Scotland, UK, 2004.

[8] G. Robles, S. Koch, and J. M. Gonzalez-Barahona. Remote analysis and measurement of libre software systems by means of the cvsanaly tool. In Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), 26th International Conference on Software Engineering, Edinburg, Scotland, UK, 2004.

[9] G. Robles-Martinez, J. M. Gonzalez-Barahona, J. Centeno-Gonzalez, V. Matellan-Olivera, and L. Rodero-Merino. Studying the evolution of libre software projects using publicly available data. In Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering, pages 111–115, Portland, Oregon, 2003.

7. Appendix: File extensions

In this appendix we will focus on the methodological part that is related to the identification of the file type as it is the most important for the goals of this paper. As it was mentioned before, the CVS logs are retrieved and parsed. We have included a procedure into CVSAnalY that enables the identification a file type by the inspection of its file name and specially by its extension. Hence, we've built a list of most common extensions and file names and later have gruoped them in several sets.

Table 3 is a small excerpt of the grouping that has been created. As it can be noted,

Table 3. Summary of file extension groups

.c, .cpp, .java, .h, .py...	Development file extensions
readme*, changelog*...	Development documentation
configure*, makefile*...	Building, compiling, conf...
.html, .txt, .pdf, .xml...	Documentation, web pages
.png, .jpg, .gif...	Images and graphics
.po, .pot, .mo...	i18n and l12n
.desktop, .ui, .xpm...	User interface
.mp3, .ogg, .wav...	Multimedia

There are some drawbacks in our classification method. The first and obvious one is that this is a heuristical procedure and hence cannot be proven to be exact in any case. Second we could mention that the heuristics could be enhanced in a simple way by looking at the content of the file for a given set of patterns that certify that the classification is correct or not. This is also a reasonable sugerence as we are working with source code that is in fact available.

Besides, there are a set of file extensions that are hardly to classify in an accurate way. This is the case for instance for HTML and text files. Usually these types of files contain information targeted to humans, although it is difficult to assess if the target group are developers (in the wide sense including also those who don't contribute code), users or just new-comers. We've decided to group all these pages in a set called "Documentation and web pages" (shortly documentation).

Files that we find that usually are tied to the development process have been grouped in a different set called "Development documentation", which includes files such as README, TODO, ChangeLog, HOWTO, etc. etc. On the other hand, in the study shown in this paper all development categories (development file extensions, development documentation and building, compiling, configuration, etc.) have been grouped into a unique set called generically "development".

Another case of uncertainty is the one related to images. Web pages usually make use of them, so they could have been classified into the documentation category. We have seen in the case study in sections 3, 4 and 5 that there is a big correlation in projects and developers among these two file types. But there are images related to other means as application design, etc. Generally, our decision has been to put images and graphcis in the "Images" set, with the exception of very clear cases as the images with the ".xpm" extension that can be classified into the user interface set without trouble.

File type identification and grouping has been tested with several huge CVS repositories and the precentage of files that cannot be classified (and that has been labeled as 'unknown') lies under 5%. Further investigation of the repository allows to identify project-own file extensions and conventions which in some cases have lowered the unknown fraction under the 3% barrier. In any case, a detailed study is pending about the amount of false positives (those files that are wrongly assigned to a given set) that this method arises, although the manual audit we have done points out that this should be not a severe deficiency of the methodology.