Using software trails to rebuild the evolution of software

Daniel German
Department of Computer Science
University of Victoria
dmgerman@uvic.ca

Abstract

This paper describes a method to recover the evolution of a software system using its *software trails*: information left behind by the contributors to the development process of the product, such as mailing lists, Web sites, version control logs, software releases, documentation, and the source code. This paper demonstrates the use of this method by recovering the evolution of Ximian Evolution, a mail client for Unix. By extracting useful facts stored in these software trails, and correlating them, it was possible to provide a detailed view of the history of this project.

1 Introduction

Investigating and recovering the evolution of a software project requires a combination of skills: it is necessary to understand the software product, its features, its components and how these have evolved; it is necessary to find, recover, and catalog valuable facts about the history of the project; it is also required to look at the developing team, in order to better understand the software process they have used, their interrelations and communication, their decision taking and their skills; it also required to start piecing together all this information, proposing potential hypothesis that are then proved or disproved. One can equate the work of a "software evolutionist" to the morphing of a software architect, a historian, an ethnologist, an anthropologist, a paleontologist, and a private investigator.

In rare cases, the evolution of a software product is recorded by an insider. This software evolutionist has access, presumably, to all the personnel and available information, and has the potential to accurately record its history as it unfolds. Unfortunately few software projects have this type of resident historian, and it is usually an outsider who has to the work. This external software evolutionist could track the project for some time, looking from the outside at how the project continually evolves. Sometimes her work is done post-mortem, looking at the remnants of the project, like an anthropologist looking for clues of how an ancient civilization functioned.

An outsider evolutionist depends on the available information of a project to tell its history. This paper defines "software trails" as pieces of information left behind by the contributors of a software project. Examples of software trails are configuration management systems logs, email messages, documentation, recordings of conversations, product releases, and of course, the source code and other required files themselves.

Software trails have the potential of keeping a "community" memory of the software development. Linus Torvalds, for example, has repeatedly said that he would not have a telephone conversation to discuss the development of the Linux kernel, because he wants every decision to be recorded for posterity. The open source community recognizes that, given the volatility of core developers and an unforeseen future, keeping this information can provide important facts critical for the long term survival of a project, taking it from one set of developers to the next one and from one maintainer to another. Arguably, the best open source success stories tend to keep very detailed software trails. These trails can be used for two purposes: to educate future developers on the characteristics of the product, and to assist in recovering the history and evolution of the product. Closed source software projects are also interested in keeping this memory, as they know that their developing teams evolve with time and they cannot be dependent on one person maintaining this information in her head.

From the point of view of the software evolutionist, the software trails left behind by a project are a gold mine, ready to be exploited. This comes at a cost: the amount of information available can be overwhelming. It is necessary to assist the software evolutionist in the process of recovering, cataloging and correlating the information available, in order to help her look at the "big picture", but at the same time, provide enough detail about a particular event in the history of a project.

2 Recovering the evolution of a project from its software trails

This paper proposes a method of recovering the evolution of a software system by analyzing the software trails left behind during its development. For example, how the version control management system logs, messages in its electronic mailing lists and the defect control system logs can be correlated to retrieve important decisions and events, and to trace how the software has evolved since its conception. This method is composed of four steps:

1. **Define Schema**: Create a schema that represents the information available in the software trails, including any relationships between them. For example, that a developer has a list of different email addresses used to post to the mailing lists; that a particular defect was fixed by a given developer with a given set of software changes; that a software change includes a delta of the change, and a version number; etc.

2. **Gather Software Trails**: Retrieve the available software trails and map them into this schema. Often the logs of these trails are not easy to parse nor translate. In some cases heuristics need to be developed and applied.

3. **Extend Information**: The available software trails can be extended by further analysis, enhancing them by extracting new facts or creating new relations as it is found appropriate. For example, many open source developers

do not use configuration management software, and the developer usually states informally (in the version control log) that a given set of changes corresponds to a defect fix; the version control log has to be parsed in order to find something that "might look" like its corresponding defect number.

4. **Analyze**: The final step is to look through this data and try to find interesting events in its development that can tell the history of the project. This is a difficult problem. As the software evolves and grows bigger, its available information grows at the same time, making it difficult for the evolutionist to find the "more" relevant information that tells an interesting fact about the history of the project.

Given the informal nature of some of this trails (and the fact that it is a reverse engineering process, where the evolutionist has no certainty on what were the actual events, and she is just merely trying to reconstruct them from the trails available) the experience and insight of the evolutionist and amount of time that she invests in the analysis of the information available will have an important impact in the quality of the results.

In order to demonstrate the above methodology, this paper uses the software trails of Ximian Evolution to recover its evolution. Evolution is a mail client (similar in scope to Microsoft Outlook) that is starting to gain popularity in the Unix world (Ximian was bought by Novell in August, 2003). Evolution developers have left software trails in mailing lists, Web sites, its CVS repository logs (CVS is one the most widely used version control systems), documentation, inside and outside the code, and Bugzilla, its bug tracking system.

3 Methodology

The following software trails were used in the recovery of the evolution of Evolution:

- Version source code releases. As of May, 2003 there have been 37 different releases. These come in the form of tar files that contain all the necessary files to build and run the product. They are made available for the people who are interested in recompiling the product to suit their particular installation. Five of these releases are considered major (0.0, 1.0, 1.1.1, 1.2.0, and 1.3.1). Evolution has adopted a numbering scheme similar to the Linux kernel, using odd numbers in the second component of a release label (such as 1, or 3 in 1.1.1 and 1.3.1 respectively) to denote "unstable" releases that are considered to be riskier (buggier) than the stable ones (like 1.0 and 1.2.0). Source code releases can be seen as a coarse grain view of the evolution of a project. A collection of scripts and tools such as "exuberant ctags" and "stripcmt" were used for fact extraction (similar to the fact extraction in [GT00]).

- CVS logs. CVS keeps track of who modifies which file, and the corresponding delta associated with the modification. This change is known as a "file revision". CVS keeps information such as who made the revision, when the actual diff of the revision, number of lines added, and number of lines removed. softChange [GM03] was used to recover the information from these logs and to enhance it. For instance, CVS does not keep track of which files are modified at the same time. softChange analyses the logs, and rebuilds these groups of files, which are then called Modification Requests (MRs). A modification request is a request by a contributor to commit a group of files at the same time. The belief is that if two files are part of the same MR, it is because they are somehow interrelated. Contrary to source code releases, CVS logs provide a very fine grained view to the evolution of the project. A snapshot of the CVS log was taken on May 21, 2003.

- Mailing lists. Evolution maintains at least two mailing list, but some of the information related to it can also be found in the GNOME mailing lists. GNOME (GNU Network Object Model Environment) is a free software collection of libraries and end-user applications that provide a graphical "desktop" for Unix systems. The project started in 1996 as a volunteer effort, and has evolved into a large system that involves paid and non-paid contributors, and it is currently shipped with almost every Linux distribution. GNOME is the parent project of Evolution. Mailing lists tend to serve as a record of important decisions related to a project. Another use of mailing lists is to announce the availability of new releases (including a summary of the new features found in it).

- ChangeLogs. The main source of documentation is the ChangeLog. As the GNU ChangeLog standards indicate, the ChangeLog explains how earlier versions of software were different from the current version.

For the purpose of this paper, softChange was extended to generate relational data, which was then imported into a `postgresql` database (the dump of the database measures 0.5 Gbytes, although some tables contain redundant information to help speed up queries –a copy of the database is available on request and it is available for download at `view.cs.uvic.ca/evolution`, along with the rest of the data used in this analysis). The analysis of the data was done ad-hoc, writing SQL queries. The results of these queries were then plotted using gnuplot.

4 Evolution

During the beginning of 1999, Bertrand Guiheneuf started working on a new mail client for the GNOME project [Gui00]. One of his goals was to create a better mail client than Balsa (then GNOME mail client) and to use Bonobo (GNOME CORBA implementation) to display the different content types in email messages. He decided to start the project by implementing a mail storage library, which he called *camel*. In Guiheneuf's view, Balsa was not good enough. He planned, however, to phase in the development of camel by incorporating its storage library into Balsa (and other potential mail clients) using CORBA [Gui99].

The *GNOME Mailer* project was formally started in April 16, 1999 with a mail message from the GNOME project leader Miguel de Icaza that discussed the need for a more powerful mail client [dI99a]. One important issue that de Icaza addressed in this message was why not to further develop an already started project (such as Balsa). His answer was "there is too much baggage in existing mail applications that we do not want to carry into the future". This message was probably triggered by Guiheneuf's posting (two weeks before). de Icaza proceeded to outline the main architecture that this client should have (which was further refined in [dI99b]):

- **Storage**. This module was to be composed of two parts: a) it will include a library to understand and interact with a variety of mail storage formats and sending email protocols (imap, pop, spool mail, UNIX mailbox, MH); and b) contain a query engine to filter, move and delete mail.
- The **Folder and Summary Display** would be the main GUI to email messages and folders.
- The **Message Display** would be responsible for displaying a particular mail message.
- The **Message Composition** would implement an editor that would allow the user to create and edit mail messages.
- Interface with the **calendar** and **addressbook** (which in de Icaza's opinion needed to be redesigned).

De Icaza, following Guiheneuf's ideas (and the trend of GNOME in general), proposed to use CORBA for communication between these modules and other applications that would help display different content types in the message display (at the time, there was a move towards making most GNOME applications CORBA aware). This module list would also serve as a way to divide the work into pieces in which different developers could concentrate and work as independently as possible. A mailing list was created for the project, and during the month of April 1999 more than 500 messages were exchanged, most of them related to requirements analysis for the new project.

Guiheneuf would become the first maintainer of the new GNOME Mailer, continuing the development of camel as its storage module. In August 1999, the name Evolution was proposed by him, and it was quickly accepted by the GNOME community[1].

In October 1999, Miguel de Icaza created Helixcode (now Ximian), a commercial venture aimed at continuing the development of GNOME, planning to generate income by selling services around it. Ximian proceeded to take under its wing the development of Evolution and has committed several employees to work on it.

In 4 years Evolution has grown into a powerful product that is starting to be widely used in the open source community. Evolution recently received the "2003 LinuxWorld Open Source Product Excellence Award" in the category of "Best Front Office Solution". One of the objectives of Evolution is to provide a free software product with functionality similar to Microsoft Outlook or Lotus Notes[Per01]. Table 1 lists the main events in the history of the project.

[1]Guiheneuf proposed e-volution, which was quickly altered to evolution. The name was later changed to Evolution and finally to its current official name Ximian Evolution.

Milestones	Date
Coding of camel starts	1999-01-01
Evolution starts	1999-04-16
Ximian is established	1999-10-01
Version 0.0	2000-05-10
Version 1.0	2001-11-21
Version 1.1.1	2002-09-09
Version 1.2.0	2002-11-07
LinuxWorld "Best Front Office Solution" award	2003-01-23
Version 1.3.1	2003-02-28

Table 1: Main milestones of the project

4.1 Releases

Figure 1 shows the growth in the size of the source code releases of Evolution. It was discovered that the total size of the release (sum of the size of all files) and the total size of the source code (sum of the size of all source code files) did not show a clear correlation. Further investigation demonstrated that the main culprit for the increase of the size of the release is its internationalization (translation files with extensions .po and .gmo). The latest version, for example, totals 64 MBytes of which 37 Mbytes (57%) are internationalization files, compared to only 11 Mbytes of source code (17%). Evolution is currently translated into 34 different languages (this does not include regional variants; for example, Evolution includes internationalization files for Portuguese and its Brazilian variant). Another surprise is to discover that the next largest contributor to the size of a release is ChangeLogs: 4.6 Mbytes (7%). ChangeLogs will be revisited in section 4.3.2.

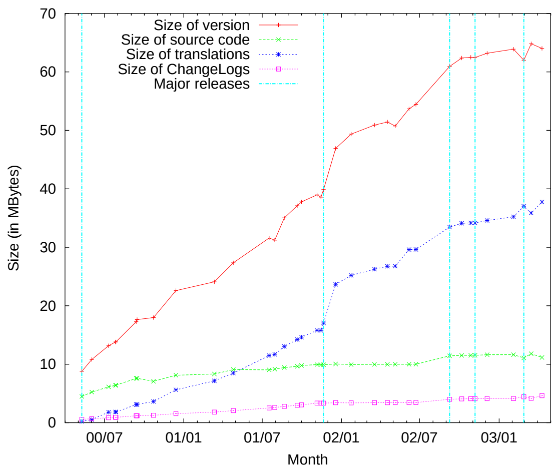

Figure 1: Size of releases over time. The plot shows also the total sizes (in Mbytes) for source code, internationalization files, and ChangeLogs. Together these 3 types of files account for more than 80% of the size of the latest version.

The number of files shows a different picture. The average proportion of source files in the releases is 46% (6.16 std deviation). In contrast, the proportion of translation files is 2.7% (0.29 stddev), and 1.1% (0.03 stddev) for ChangeLogs. Translation files and ChangeLogs are therefore few, but very large, when compared to source code files.

Figure 2 shows, for a given release, the number of source code files, total LOCS and total cleanLOCS (number of LOCS when comments and empty lines have been removed). The average size of a source file has been stable across versions, at 639 (25 stddev) LOCS per .c and 101 (7.6 stddev) LOCS per .h file. The proportion of cleanLOCS to LOCS has also remained stable across versions, at 72.5% (1.4 stddev) for .c files, and 60% (2.6 stddev) for .h files.

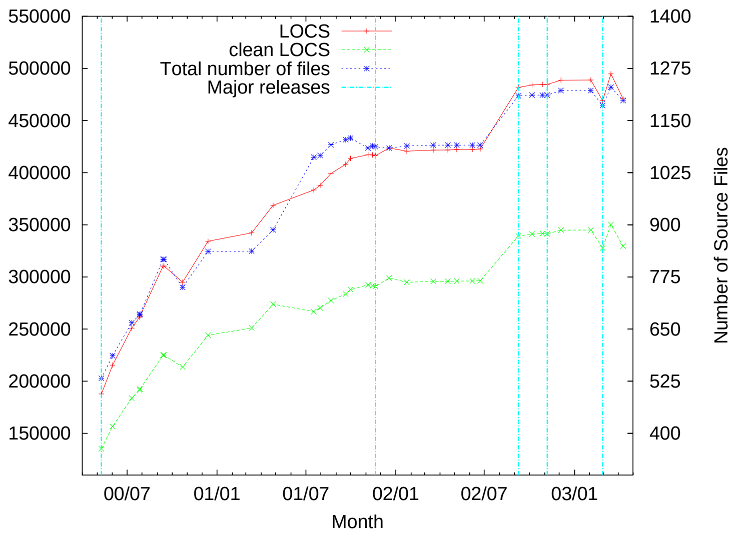

Figure 2: LOCS of releases over time. The plot shows the total number of files, and the number of clean LOCS (LOCS without comments nor empty lines).

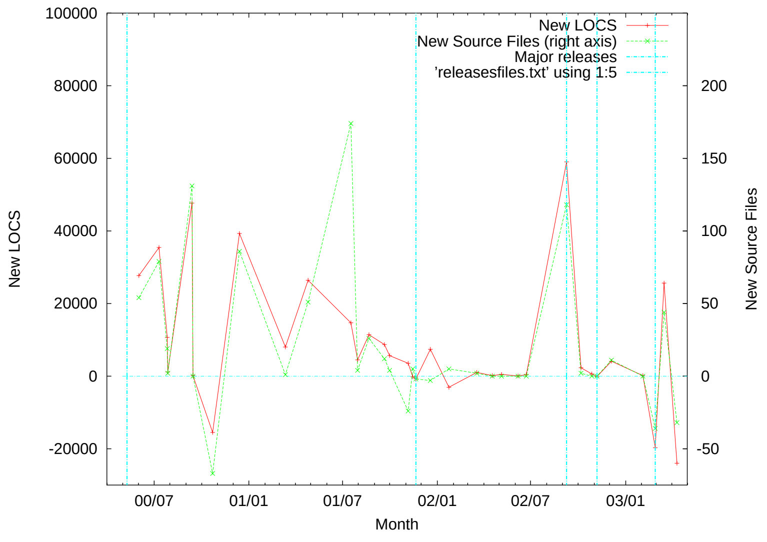

Figure 3: Changes in LOCS and number of files, per version

The actual change in LOCS from one version to another show an interesting story. Figure 3 shows the increment in the LOCS and number of files over time. Of special interest are the negative increments in either LOCS or source files, suggesting removal of source code. For example, in version 0.6 (released 2000/10/23) 15.5 kLOCS and 67 source code files were removed with respect to the previous version (0.5.1). Between these two releases 157 source code files were deleted and 90 created (45 kLOCS were deleted and 24k LOCS added). Further analysis of the available software trails showed that for this release it was decided to move several widgets (from Evolution's GUI) to the Gal project. Gal, according to its official description is "the GNOME Application Library, a collection of widgets and other helper functions originally extracted from Evolution and gnumeric (GNOME spreedsheet)". In fact, the first version of Gal (0.1) was released in Oct. 5, 2000 [dI00], 5 days before Evolution 0.6 (and the sudden drop in LOCS).

4.2 Development Activity

One important question that arises when looking at the increment in the size of Evolution is how does it correlate to the actual activity of the developers? The CVS logs provides some useful information that can be used to attempt to answer this question.

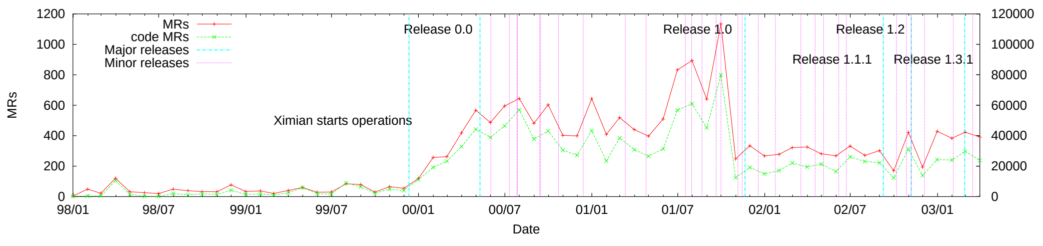

Figure 4: Evolution of the project in number of MRs

Figure 4 shows the number of MRs per month for Evolution. The plot also shows the different releases in the project. There are several interesting observations from this graph. First, the development activity was relatively flat during the first year of the development, and it is not until Ximian is born that there is a surge in the number of MRs. The number of MRs surges just before release 1.0. After that, the number of MRs remains more stable, but still shows peaks that correspond to releases. Because it is not possible to have access to the actual number of hours spent per developer in the project, it is not possible to determine the development effort spent per MRs, and therefore, if less MRs mean less developer-time, or if some MRs required more time. In the same figure, the number of MRs that involve source code (codeMRs) is also shown. The proportion of codeMRs to MRs has decreased during 2003 (approximately 38% of the MRs do not involve source code).

Why has the proportion of codeMRs dropped? The exploration of the logs drew the following conclusions. From all MRs in 2003, 86% corresponded to changes in source code (61%), translations (13%) and changes to metafiles (files with extension .am and .in, 18%)[2]. Metafiles are used by the automake and autoconf tools to

[2]Some MRs included changes to Metafiles and source code, and some MRs included changes to metafiles and translations

create other files. The most common use of these Metafiles is the creation of Makefiles (the developer creates an .am or .in file, and autoconf and automake create the corresponding Makefile). Metafiles rely heavily on macros (GNOME provides a module called macros with the majority of these definitions).

A surge in the activity related to Metafiles and translations was to blame for the drop in the proportion of codeMRs. The question that followed was, what prompted the surge in Metafile activity? In those MRs 70% of the revisions corresponded to Makefile.am files; and 12% of the revisions corresponded to changes to `configure.in`, the main autoconf file that drives the configuration of Evolution when a user wants to compile it. Inspection of the ChangeLogs seems to suggest a conscious effort to cleanup the Metafiles.

The surge in changes to the translations is attributed to a previous significant change in the UI. Once the development team decides to make a "freeze" in the features of a release, translators start making changes to the corresponding translation files.

Another question prompted by figure 4 is why does it show activity before January, 1999? It appears that some code that was in development previous to Evolution was later incorporated into it (one widget and some calendar related code). It is also suspected that some revisions contain invalid dates, suggesting that during a period of time the machine's clock was set to an incorrect time.

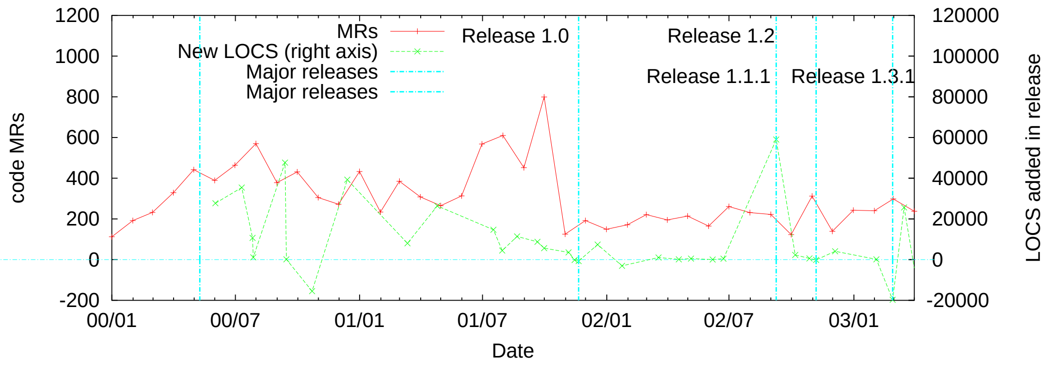

Figure 5: Changes in LOCS and number of files, per version

Figure 5 shows codeMRs and how they relate to the actual growth in the size of the source code in the releases. Even in periods where the code base does not increase (like the first half of 2002) the number of MRs is still large. This suggests a period in which debugging took precedence over development of new features.

4.2.1 Characteristics of MRs

It is also interesting to see the typical characteristics of an MR. Figure 6 shows the number of files per MR. Most of them contain a small number of files, which is a healthy sign. The log for the largest MR (which contains 650 files, in 2001/06/23) reads "Update the copyrights, replacing Helix Code with Ximian and helixcode.com with ximian.com all over the place.". That day a total of 709 files were modified. Similarly, the largest number of files modified in a single day was 1417 (2001/10/27) and the reason was "update the licensing information to require version 2 of the GPL (instead of version 2 or any later version)". These two explanations highlight a particular feature of MRs in Evolution: developers take good care of explaining in each MR the reason for the change (CVS allows developers to add a log message to each MR). The average log for an MR is 300 characters (561 stddev, 170 median), with a minimum length of 1 (only 8 MRs) and 18K for the longest log (which involved the merging of a branch to the main CVS tree).

From a total of 18K MRs, only 87% include two or more files in it. A preliminary analysis shows that most of these MRs are of two types: a) files which were overlooked in a previous MR and committed minutes later; and b) minor corrections, such as fixing spelling mistakes. Further analysis is needed to corroborate this hypothesis.

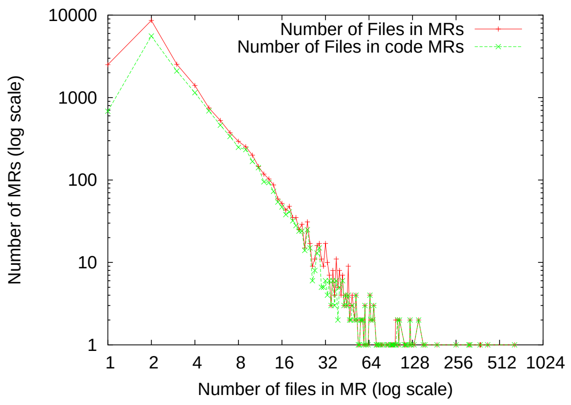

Figure 6: Most MRs contain a small number of files

4.2.2 Contributors

There is a common belief that open source projects are developed by a large number of individuals. While that is true, it is important to recognize that the contribution of the majority of these individuals is very small. In open source projects, contributors can be divided into two main groups: those with write access to the CVS repository (and can make their contributions to the CVS repository themselves) and those who do not have write access to the repository. In GNOME it is not difficult to get write access to the repository. Once somebody has submitted several contributions, this person can apply for CVS write access. In GNOME, more than 500 people have CVS write access[3].

By looking at the changes committed by contributors with CVS write access, we can see that like many other open source projects, the majority of the coding is done by a small number of individuals. Zawinsky, at one time one of the core Mozilla contributors, commented on this phenomenon: "If you have a project that has five people who write 80% of the code, and a hundred people who have contributed bug fixes or a few hundred lines of code here and there, is that a 105-programmer project?"—as cited in [Jon02].

Evolution contains contributions by 201 different userids (to which, this paper will refer as contributors). Few of these, however, contributes a significant portion

[3]The author has write access to the GNOME CVS repository.

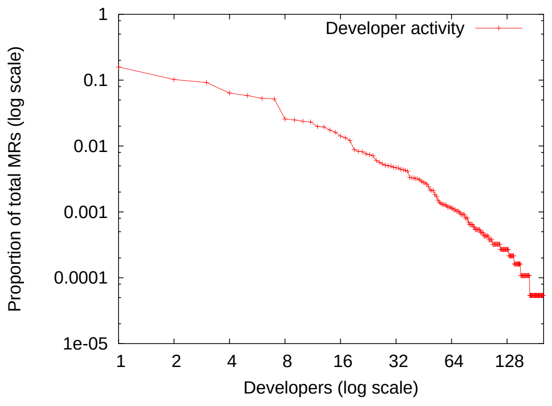

Figure 7: Proportion of MRs per contributor. Each contributor was assigned a number from 1 to 201, which corresponds to the X axis.

of the MRs. Figure 7 shows the proportion of MRs per contributor (each contributor was assigned a number from 1 to 201, which corresponds to the X axis). Only 18 contributors accounted each for more than 1% of the total MRs. The largest contributor is responsible for 16% of the MRs, while at the other side of the spectrum 32 contributors had only one MR only. Furthermore, a total of 48% of the MRs were contributed by only 5 contributors, while 142 contributors contributed just 5% of the MRs (80 contributed a total of 1% of the MRs).

Table 2 shows the 11 most active developers, as a proportion of all MRs. The top 10 appear to be Ximian employees or consultants (see http://primates.ximian.com/). This fact corroborates the hypothesis that private companies (such as RedHat, Ximian, and Eazel) have had a very important effect on the development of the GNOME project [Ger02]. In that respect it is similar to the Mozilla project where core contributors were employees of Netscape (see [MFH02]).

Userid	**Prop.**	**Accum.**
fejj	0.16	0.16
ettore	0.10	0.26
danw	0.09	0.35
zucchi	0.06	0.42
clahey	0.06	0.48
jpr	0.05	0.53
toshok	0.05	0.58
federico	0.03	0.61
peterw	0.02	0.63
iain	0.02	0.65
other	0.35	1.00

Table 2: Most active developers, as a proportion of total MRs

How regularly were contributors participating in the project? The number of different contributors by year is depicted in table 3. After January 2000, in any given month there is an average of 32 contributors (8.3 stddev, minimum 15, maximum 47) per month to the project.

Year	**Number of Contributors**
1998	37
1999	54
2000	95
2001	98
2002	79
2003	56

Table 3: Contributors to the project by year. It takes into account only those contributors with CVS write access.

4.3 Revisions

Every time a file is modified, CVS creates a record of who modifies it, when, and the "delta" of the modification. This modification is known in CVS lingo as a "revision".

4.3.1 Types of Files

Extension	**Prop.**	**Accum.**	**Number of files in CVS**
.c	0.41	0.41	1195
ChangeLog	0.22	0.62	43
.h	0.13	0.75	1063
.am	0.05	0.81	174
.po	0.04	0.85	71
.ics	0.02	0.87	396
.sgml	0.02	0.90	228
.in	0.02	0.92	136
.png	0.01	0.93	405
other	0.07	1.00	

Table 4: Revisions and number of files per file extension. C files (.h and .c) and ChangeLog modifications account for 75% of total revisions.

Table 4 shows the proportion of revisions per extension (i.e. type of file) and it tells an interesting story. Given that C is the language of choice for Evolution, it is not surprising to see .c and .h files at the top, along with ChangeLogs (ChangeLogs are discussed in detail in section 4.3.2). Metafiles (.am and .in) and translations follow. The next file extension .ics corresponds to files that include information about a particular location in the world, particularly its time zone. There were 1903 revisions made to 396 .ics files, for an average of 4.8 changes per file.

Files with extension .sgml are documentation files. As with many open source projects, the documentation is written in SGML using the docbook DTD. Finally, .png files correspond to artwork.

4.3.2 ChangeLogs

ChangeLog files are an important source of information about the development and evolution of a project. The Evolution developers are fairly consistent in their modifications to the ChangeLog files. From all MRs involving 2 or more files, 93% include a modification to a ChangeLog. Evolution developers seem to make sure that they document their changes in the corresponding ChangeLogs. Table 5 shows the 10 most modified files, 8 of them are ChangeLogs. ChangeLogs (and CVS logs) can provide insight on patches submitted by developers without a CVS account, as developers are expected to be careful to give credit to the patch submitter in the corresponding ChangeLog entry (which are not taken into account for this paper).

4.3.3 Source Code Hot Spots

There have been a total of 41120 revisions to 2258 source code files[4]. Figure 8 shows the proportion of revisions per source code file. 51 files account for 25% of the total number of revisions, while 764 account for only 5% of them.

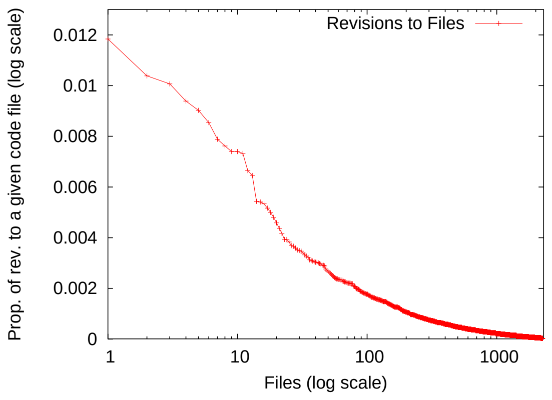

Figure 8: Proportion of revisions per source code file. Each file was assigned a number from 1 to 2258, which corresponds to the X axis.

4.4 Modularization

The success of an open source project depends on the ability of its maintainers to divide it into small parts in which contributors can work with minimal communication between each other and with minimal impact on the work of others [LT00]. From the beginning of the project, there has been a conscious attempt to divide Evolution into modules that fulfill the previous characteristics. Modules are represented in the code base as subdirectories. Figure 9 shows the different modules and the number of MRs for each of them, representing the level of activity in each module.

Figure 10 shows the size of the seven largest modules in Evolution in terms of LOCS. With the exception of libical and widgets, modules tend to grow in size. Before 2002, both libical and widgets show a lot of variability in both their sizes

[4]Many of these files are no longer in the latest release, as they have been removed during the development process. Nonetheless, CVS keeps information about their modification.

File	**Prop.**	**Accum.**
mail/ChangeLog	0.04	0.04
calendar/ChangeLog	0.03	0.06
camel/ChangeLog	0.03	0.09
addressbook/ChangeLog	0.02	0.11
shell/ChangeLog	0.02	0.13
ChangeLog	0.02	0.14
po/ChangeLog	0.02	0.16
configure.in	0.01	0.17
composer/ChangeLog	0.01	0.18
mail/mail/callbacks.c	0.01	0.18

Table 5: Top 10 most modified files. ChangeLogs clearly take the lead. As its name implies, mail-callbacks.c contains the callbacks of the mail client, hence the frequency at which it is modified. These 10 files account for a total of 18% of all file revisions.

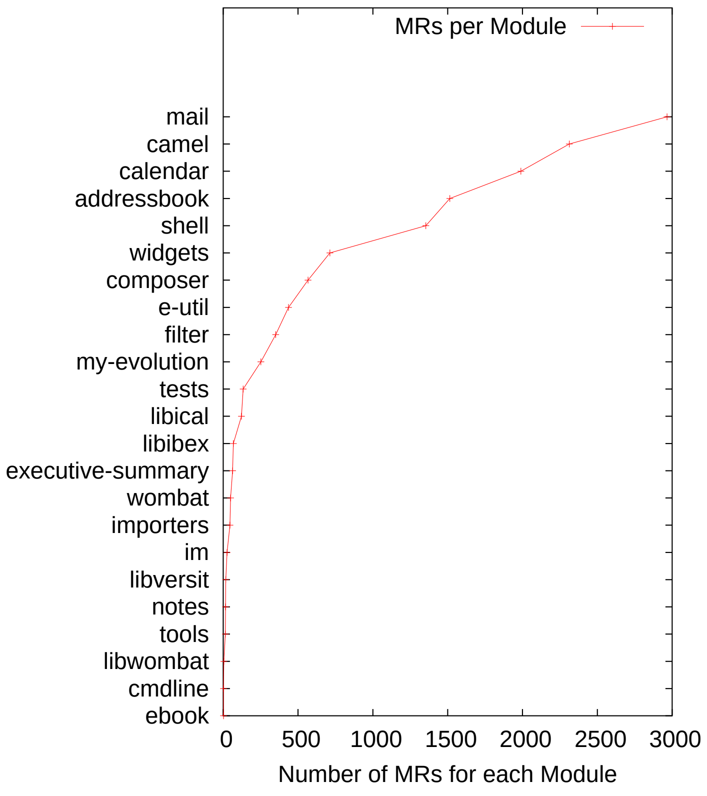

Figure 9: MRs per module. Most of the activity is concentrated in few modules.

and the number of files in them. After Version 1.0, the size of Evolution has been growing at a very small pace.

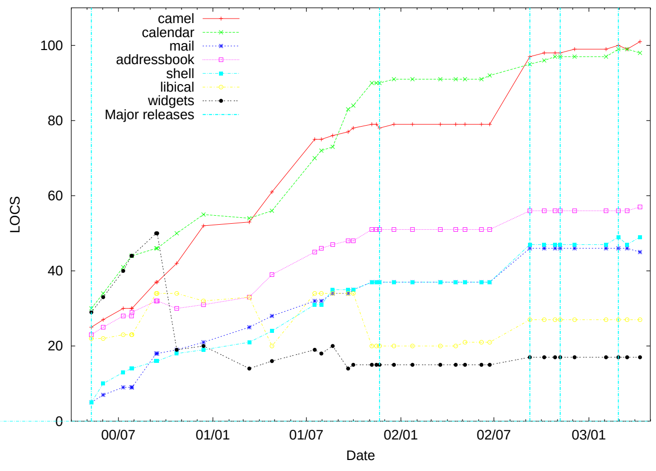

Figure 10: LOCS in selected modules, per version

Other interesting questions are: Do contributors tend to concentrate in one module? How many core contributors does a given module have? Table 6 shows that information for the five most active modules of Evolution. In order to account only for people who are still active in the development, this table only shows data related to MRs which happened in 2002. It is not surprising to see that one or two contributors are responsible for at least two thirds of the MRs in each module.

Finally, how well do modules isolate developers from the complexity of other modules? One potential way to measure this dependency is to analyze the number of codeMRs that require changes in more than one module. Figure 11 shows a compelling story: only 3% of the MRs include more than one module. Further analysis of the changes is required to determine what is the proportion of changes that were actual code changes compared to changes in comments (such as the change in license, described in section 4.2.1).

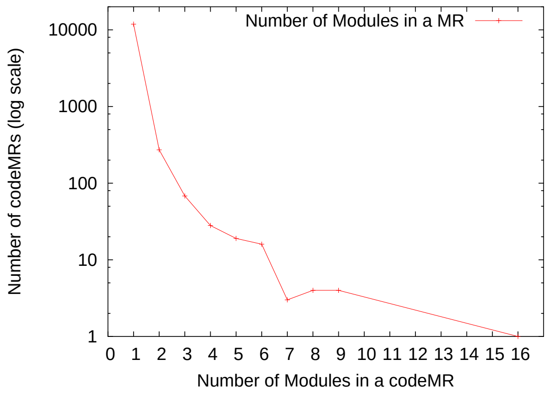

Figure 11: Number of different modules that appear in a codeMRs. The proportion of codeMRs that involve more than one module is very small (3%).

Mod	**Progs**	**Id**	**Prop**	**Acc**
shell	17	ettore	0.65	0.65
		danw	0.11	0.76
		toshok	0.05	0.81
		clahey	0.04	0.84
		zucchi	0.03	0.87
mail	19	fejj	0.52	0.52
		rodo	0.13	0.65
		zucchi	0.12	0.77
		ettore	0.07	0.83
		danw	0.06	0.89
calendar	17	jpr	0.40	0.40
		rodrigo	0.32	0.72
		ettore	0.07	0.79
		danw	0.06	0.85
		damon	0.03	0.88
camel	9	fejj	0.66	0.66
		zucchi	0.25	0.91
		danw	0.03	0.94
		peterw	0.03	0.97
		ettore	0.01	0.99
addressbook	19	toshok	0.57	0.57
		clahey	0.13	0.70
		ettore	0.09	0.79
		danw	0.07	0.87
		fejj	0.03	0.90

Table 6: Top 5 programmers of some the most active modules during 2002. The first column shows the name of the module, the second shows the total number of programmers who contributed to it in that year, the third shows the userid of the top 5 programmers and the proportion of their MRs with respect to the total during the year.

5 Further observations

The results described in this paper show that the method described in section 2 can be applied to recover the evolution of a software project where the amount of software trails is significant. Several observations can be made about this experience.

- One software trail does not tell the whole story. It is paramount to cross-reference software trails to really understand what they mean in the evolution of the project. For example, the size of software releases in Evolution has been growing in linear fashion, while the growth in the size of the source code is relatively flat; also, many developers have been participating in the project, but most of them with very few contributions.

- Schema definition. The schema used in this study kept changing, in part due to the incorporation of new trails, and in part because new information and relations kept being discovered. It is expected that, as this type of analysis becomes more pervasive, standard schemas can be developed. This will have 2 advantages: a) it will promote the creation of tools that gather software trails, extend them, and analyze them; and b) the evolutionist will better understand the nature and interrelation of the available trails before starting to do her work.

- One of the main challenges of analyzing software trails is that many of them are informal in nature. For example, email messages contain a large amount of information pertaining to the way the project has evolved, but they are difficult to analyze in an automatic fashion. Correlating different trails is also an error prone task, in which heuristics have to be developed and tested. It might be the case that a heuristic performs differently in different projects.

- Information overload and the need for analysis and visualization tools. The amount of available information makes it indispensable to use tools that can filter it and visualize it. Again, as schemas are standardized, different research teams could provide different tools that specialize in mining and visualizing certain types of trails. In this paper, SQL was chosen because it provides a sophisticated query language (further extended in postgreSQL with its support for regular expressions in the `where` clause). SQL was very helpful in filtering and tabulating information, that could then be plotted (our research team has since developed a tool to automatically create many of the plots displayed herein using SVG using the Web as its interface). It is also interesting that Evolution itself proved very useful in analyzing the Evolution mailing lists, given that it provides a powerful query language for email messages.

- Quality of software trails. It is important to state that not all development teams generate "good" software trails. In the experience of the author, there is a point in a software project in which software trails start to "mature" and this point is likely a correlation of the success of the project, the level of interaction that developers have to have, and their maturity, and in the case of commercial projects, the influence of management. For instance, there is very little information about Evolution when only one developer was contributing to it, but as the developers grew in number (and became more experienced) their trails improved in quality. The Free Software Foundation has an important effect in the quality of trails, as it publishes a collection of guidelines that free software developers should follow.

6 Conclusions and Future Work

This paper demonstrated a methodology to recover the evolution of a software project using its software trails. Software trails, such as version releases, version control logs and mailing lists were used to recover the evolution of Ximian Evolution, a free (as defined by the GPL) mail client for Unix. The analysis of these software trails allowed the discovery of interesting facts about the history of the project: its growth, the interaction between its contributors, the frequency and size of the contributions, and important milestones in its development.

There are several potential avenues for future research. One of them is to create tools that analyze and enhance the facts extracted. For example, CVS's MRs can be analyzed in an attempt to guess the type of modification that the developer intended: a comment, a bug fix, a new feature, or refactoring, for example. This will allow the evolutionist to quickly categorize changes and concentrate on those of interest.

Another area of research is the visualization of this information. As the project grows older, its trails grow in number. It is necessary to create tools that analyze and display the gathered facts to the user and allow its visualization in a highly dynamic manner. Metrics are also an important area of research. It is needed to quantify the information extracted from software trails, so it can be compared with other software projects. For example, how can the "disjointness" of contributors of different modules to different software projects be quantified and compared?

Finally, studies on other software projects (similar to the one done in this paper) are needed. These studies will provide information necessary to better understand the characterization of software trails. Furthermore, these studies will allow researchers to compare the evolution of different software projects; and to a certain extend some of the practices used by their corresponding development teams.

Acknowledments

This research has been supported by the National Sciences and Engineering Research Council of Canada, and the British Columbia Advanced Systems Institute. The author would like to thank Audris Mockus (co-author of softChange) for his invaluable help in a preliminary analysis of Evolution and the reviewers of this paper for their helpful comments.

References

[dI99a] Miguel de Icaza. Writing a GNOME mail client. http://mail.gnome.org/archives/gnome-announce-list/1999-April/msg00029.html, April 1999.

[dI99b] Miguel de Icaza. Writing a GNOME mail client. http://canvas.gnome.org:65348/mailing-lists/archives/gnome-mailer-list/1999-April/0018.shtml, April 1999.

[dI00] Miguel de Icaza. G Apps Lib 0.1 is out. http://mail.gnome.org/archives/gnome-announce-list/2000-October/msg00005.html, October 2000.

[Ger02] Daniel M. German. The evolution of the GNOME Project. In *Proceedings of the 2nd Workshop on Open Source Software Engineering*, May 2002.

[GM03] Daniel M. German and Audris Mockus. Automating the Measurement of Open Source Projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, May 2003.

[GT00] Michael W. Godfrey and Qiang Tu. Evolution in Open Source Software: A Case Study. In *Proc. of the 2000 Intl. Conference on Software Maintenance*, pages 131–142, 2000.

[Gui99] Bertrand Guiheneuf. Gnome Mail clients (Re: Is Balsa alive?). http://mail.gnome.org/archives/gnome-devel-list/1999-April/msg00042.html, April 1999.

[Gui00] Bertrand Guiheneuf. Candidate (Bertrand Guiheneuf). http://mail.gnome.org/archives/foundation-announce/2000-October/msg00009.html, Oct 2000.

[Jon02] Paul Jones. Brooks' law and open source: The more the merrier? does the open source development method defy the adage about cooks in the kitchen? IBM developerWorks, August 20, 2002.

[LT00] Josh Lerner and Jean Triole. The Simple Economics of Open Source. Working Paper 7600, National Bureau of Economic Research, March 2000.

[MFH02] Audris Mockus, Roy T. Fielding, and James Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3):1–38, July 2002.

[Per01] Ettore Perazzoli. Ximian Evolution: The GNOME Groupware Suite. http://developer.ximian.com/articles/ whitepapers/evolution/, 2001.