11 Software Engineering Practices in the GNOME Project

Daniel M. German

One of the main goals of empirical studies in software engineering is to help us understand the current practice of software development. Good empirical studies allow us to identify and exploit important practical ideas that can potentially benefit many other similar software projects. Unfortunately, most software companies consider their source code and development practices to be a trade secret. If they allow researchers to investigate these practices, it is commonly under a nondisclosure agreement. This practice poses a significant problem: how can other researchers verify the validity of a study if they have no access to the original source code and other data that was used in the project? As a consequence, it is common to find studies in which the reader is asked to trust the authors, unable to ever reproduce their results. This situation seems to contradict the main principle of empirical science, which invites challenge and different interpretations of the data, and it is through those challenges that a study strengthens its validity.

Bruce Perens describes open source software (OSS) as software that provides the following minimal rights to their users: (1) the right to make copies of the program and distribute those copies; (2) the right to have access to the software's source code; and (3) the right to make improvements to the program (Perens 1999). These rights provide empirical software engineering researchers with the ability to inspect the source code and share it without a nondisclosure agreement. Furthermore, many of these projects have an "open source approach" to the historical data of the project (email, bug tracking systems, version control). Finally, researchers can participate actively or passively in the project in a type of anthropological study.

The economic model on which closed source software (CSS) projects are based is fundamentally different than that of OSS projects, and it is necessary to understand that not all *good* OSS practices might be transferable

to the realm of CSS projects. It is, however, important to identify these practices, which at the very least will benefit other OSS projects, and at best, will benefit any software project.

As of March 2003, SourceForge.net lists more than 58,000 projects and more than 590,000 users. Even though most of these projects might be small, immature, and composed by only a handful of developers (Krishnamurthy 2002), the number of projects suggests that the study of practices in OSS software development is an important area of research with the potential to benefit a significant audience.

One free software (FS) project (and therefore OSS) of particular interest is (http://www.gnome.org) the GNU Network Object Model Environment (GNOME). GNOME is an attempt to create a free (as defined by the General Public License, or GPL) desktop environment for Unix systems. It is composed of three main components: an easy-to-use graphical user interface (GUI) environment; a collection of tools, libraries, and components to develop this environment; and an "office suite" (Gwynne 2003). There are several features of GNOME that make it attractive from the point of view of a researcher:

1. GNOME is a widely used product. It is included in almost every major Linux distribution: Sun offers GNOME for Solaris, IBM offers it for AIX, and it is also available for Apple's OS X.
2. Its latest version (2.2) is composed of more than 2 million lines of code divided into more than 60 libraries and applications.
3. More than 500 individuals have contributed to the project (those who have write-access to the CVS repository) and contributors are distributed around the world.
4. GNOME contributors maintain a large collection of information relevant to the project that traces its history: several dozens of mailings lists (including their archives), a developers, Web site with a large amount of documentation, a bug tracking system, and a CVS repository with the entire history of the project dating to 1997.
5. Several commercial companies, such as Red Hat, Sun Microsystems, and Ximian contribute a significant amount of resources to the project, including full-time employees, who are core contributors to the project. In some cases, these companies are almost completely responsible for the development of a library or an application (for an example see "Case Study: Evolution" later in this chapter). Given that most of the contributors of these projects belong to the same organization, it could be argued that they resemble CSS projects.

6. From the economic and management point of view, it provides an interesting playground in which the interests of the companies involved (who want to make money) have to be balanced with the interests of the individuals who contribute to the project (who are interested in the openness and freedom of the project).

Architecture

GNOME targets two different types of audiences. On one hand, it is intended for the final user, who is interested on having a cohesive set of applications for the desktop, including an office suite. On the other hand, it contains a collection of APIs and development tools that assist programmers in the creation of GUI applications.

The deliverables of GNOME are therefore divided into three main groups:

1. *Libraries*. One of the first goals of the project was to provide a library of GUI widgets for X11[1]. It currently contains libraries for many other purposes: printing, XML processing, audio, spell-checking, SVG and HTML rendering, among others. Any function that is expected to be used by more than one application tends to be moved into a library.
2. *Applications*. The official distribution of GNOME contains a minimal set of applications that includes a "task bar" (called a "panel" in GNOME), applets to include in the task bar, a text editor, a windows manager, a file manager, and helper applications to display a variety of file types. GNOME also includes a set of optional applications for end users, such as a mail client, a word processor, a spreadsheet, and an accounting package; and some for developers, such as an IDE (Integrated Development Environment).
3. *Documentation*. Documentation comes in two types: one for developers and one for final users. The former includes a description of the APIs provided by the libraries, while the latter describes how to use the different GNOME applications.

Figure 11.1 depicts the interaction between libraries, applications, and the rest of the operating system. Libraries provide common functionality to the applications and they interact with X11, with the operating system, and with non-GNOME libraries. GNOME applications are expected to use these libraries in order to be isolated from the running environment. The window manager (which is required to be GNOME-aware) serves also as an intermediary between GNOME applications and X11. ORBit is the

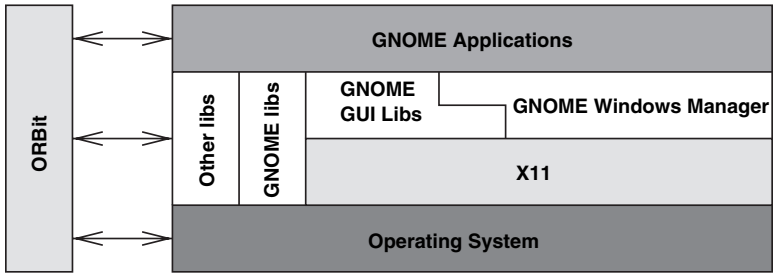

Figure 11.1
GNOME is composed of a collection of libraries and applications. The libraries are responsible for interacting between the user and X11 or the operating system. ORBit is a CORBA implementation that is responsible for interprocess communication.

GNOME implementation of CORBA and it is responsible for the communication between applications.

Project Organization

The success of an OSS project depends on the ability of its maintainers to divide it into small parts on which contributors can work with minimal communication between each other and with minimal impact to the work of others (Lerner and Triole 2000). The division of GNOME into a collection of libraries and applications provides a natural way to split the project into subprojects that are as independent as possible from each other. These subprojects are usually called *modules* (*module* is the name used by CVS to refer to a directory in the repository). The core distribution of GNOME 2.2 is composed of approximately 60 different modules.

And as a project grows bigger, it tends to be split into further submodules, which minimize the interaction between contributors. Using `softChange` to analyze the historical data from the CVS repository (German and Mockus 2003) has resulted some interesting facts about the division of work in GNOME. Because CVS does not have a notion of "transaction" (identifying an atomic operation to the repository), one of the tasks of `softChange` is to try to reconstruct these transactions (which `softChange` calls a Modification Request, or MR). The CVS data of a total of 62 modules was analyzed. In order to account for contributors that are no longer active, it was decided to narrow the analysis to 2002. An MR can be started only by a contributor who has a CVS account. Sometimes a patch is submitted, by someone who does not have a CVS account, to one of the

core contributors of the corresponding module, who then proceeds to evaluate it, accept it, and commit it if appropriate. This analysis does not take these sporadic contributors into account.

In 2002, a total of 280 people contributed to these 62 modules. It was decided to further narrow the analysis and consider only contributors to the code base (who will be referred as *programmers*) and consider only MRs involving C files (C is the most widely used language in GNOME, and the number of C files in these modules outnumber the files in the next language—bash—by approximately 50 times).

A total of 185 programmers were identified. Ninety-eight programmers contributed 10 or fewer MRs, accounting for slightly less than 3 percent of the total MRs. The most active programmer (in terms of MRs) accounted for 7 percent of the total. The top 10 programmers accounted for 46 percent of the total MRs. Even though these numbers need to be correlated with the actual number of lines of code (LOCS) or defects (bugs) removed per MR, they indicate that a small number of developers are responsible for most of the coding of the project. Zawinsky, at one time one of the core Mozilla contributors, commented on this phenomenon: "If you have a project that has 5 people who write 80 percent of the code, and 100 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 Jones 2002). When taking into account the division of the project into modules, this effect seemed more pronounced. Table 11.1 shows the top five programmers for some of the most actively modified modules of GNOME.

Module Maintainers

Module maintainers serve as leaders for their module. Lerner and Triole (2000) identified the main roles of a leader in an OSS project as:

- Providing a vision
- Dividing the project into parts in which individuals can tackle independent tasks
- Attracting developers to the project
- Keeping the project together and preventing forking[2]

GNOME has been able to attract and maintain good, trustworthy maintainers in its most important modules. Many of these maintainers are employees paid by different companies to work on GNOME.

As it was described in German 2002, several companies have been subsidizing the development of GNOME. Red Hat, Sun Microsystems, and Ximian are a few of the companies who pay full-time employees to work

Table 11.1
Top five programmers of some the most active modules during 2002

Module	Total number of programmers	Programmer	Proportion of MRs
glib	24	owen	31%
		matthiasc	18%
		wilhelmi	10%
		timj	10%
		tml	9%
gtk+	48	owen	37%
		matthiasc	12%
		tml	9%
		kristian	8%
		jrb	4%
gnome-panel	49	mmclouglin	42%
		jirka	12%
		padraigo	6%
		markmc	6%
		gman	6%
ORBit2	11	michael	51%
		mmclouglin	28%
		murrayc	9%
		cactus	5%
		scouter	3%
gnumeric	19	jody	34%
		mortenw	23%
		guelzow	17%
		jpekka	12%
		jhellan	9%

The first column shows the name of the module, the second shows the total number of programmers who contributed in that year, and the third shows the userid of the top five programmers and the proportion of their MRs with respect to the total during the year. In this table, only MRs that included C files are considered.

on GNOME. Paid employees are usually responsible for the following tasks: project design and coordination, testing, documentation, and bug fixing. These tasks are usually less attractive to volunteers. By taking care of them, the paid employees make sure that the development of GNOME continues at a steady pace. Some paid employees also take responsibility (as module maintainers) for some of the critical parts of the project, such as gtk+ and ORBit (Red Hat), the file manager Nautilus (Eazel, now bankrupt), and Evolution (Ximian). Paid employees contribute more than just code; one of the most visible contributions of Sun employees is the proposal of the GNOME Accessibility Framework a set of guidelines and APIs intended to make GNOME usable by a vast variety of users, including persons with disabilities. For example, in Evolution, the top 10 contributors (who account for almost 70% of its MRs) are all Ximian employees.

Volunteers still play a very important role in the project, and their contributions are everywhere: as maintainers and contributors to modules, as bug hunters, as documenters, as beta testers, and so on. In particular, there is one area of GNOME development that continues to be performed mainly by volunteers: internationalization. The translation of GNOME is done by small teams of volunteers (who usually speak the language in question and who are interested in support for their language in GNOME).

As with any other open source project, GNOME is a meritocracy, where people are valued by the quality (and quantity) of their contributions. Most of the paid contributors in GNOME were at some point volunteers. Their commitment to the project got them a job to continue to do what they did before as a hobby.

Requirement Analysis

Most OSS projects have a requirement's engineering phase that is very different from the one that takes place in traditional software projects (Scacchi 2002). At the beginning of GNOME the only stakeholders were the developers, who acted as users, investors, coders, testers, and documenters, among other roles. While they had little interest in the commercial success of the project, they wanted to achieve respect from their peers for their development capabilities and wanted to produce software that was used by the associated community. In particular, the following sources of requirements in GNOME can be identified (German 2003):

- *Vision.* One or several leaders provide a list of requirements that the system should satisfy. In GNOME, this is epitomized by the following nonfunctional requirement: "GNOME should be completely free software"

(*free* as defined by the Free Software Foundation; free software gives the following rights to its users: to run the software for any endeavor; to inspect its source code, modify it; and to redistribute the original product or the modified version).

- *Reference applications*. Many of its components are created with the goal of replacing similar applications. The GNOME components should have most of the same, if not the exact same, functionality as these reference applications. For example, gnumeric uses Microsoft Excel as its reference, ggv uses gv and kghostview, and Evolution uses Microsoft Outlook and Lotus Notes.
- *Asserted requirements*. In a few cases, the requirements for a module or component are born from a discussion in a mailing list. In some cases, a requirement emerges from a discussion whose original intention was not requirement analysis. In other instances (as in the case of Evolution), a person posts a clear question instigating discussion on the potential requirements that a tool or library should have. Evolution was born after several hundred messages were created describing the requirements (functional and nonfunctional) that a good mailer should have before coding started. More recently, companies such as Sun and IBM have started to create requirements documents in areas that have been overlooked in the past. One of them is the GNOME Accessibility Framework.
- *A prototype*. Many projects start with an artifact as a way to clearly state some of the requirements needed in the final application. Frequently a developer proposes a feature, implements it, and presents it to the rest of the team, which then decides on its value and chooses to accept the prototype or scrap the idea (Hissam et al. 2001). GNOME, for example, started with a prototype (version 0.1) created by Miguel de Icaza as the starting point of the project.
- *Post hoc requirements*. In this case, a feature in the final project is added to a module because a developer wants that feature and he or she is willing to do most of the work, from requirements to implementation and testing. This feature might be unknown to the rest of the development team until the author provides them with a patch, and a request to add the feature to the module.

Regardless of the method used, requirements are usually gathered and prioritize by the maintainer or maintainers of a given module and potentially the Foundation (see next section, "The GNOME Foundation"). A maintainer has the power to decide which requirements are to be implemented and in which order. The rest of the contributors could provide

input and apply pressure on the maintainers to shape their decisions (as in post hoc requirements). A subset of the contributors might not agree with the maintainer's view, and might appeal to the Foundation for a decision on the issue. These differences in opinion could potentially jeopardize the project and create a fork. So far this has not happened within GNOME. On the other hand, some contributors have left the project after irreconcilable differences with the rest of the team. For example, Carsten Haitzler Rasterman, creator of Enlightenment, left GNOME partially due to differences in opinion with the rest of the project (Haitzler 1999).

The GNOME Foundation

Until 2000, GNOME was run by a "legislature," in which each of its contributors had a voice and a vote and the developer's mailing list was the floor where the issues were discussed. Miguel de Icaza served as the constitutional monarch and supreme court of the project, and had the final say on any unsolvable disputes. This model did not scale well, and was complicated when 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.

In August 2000, the GNOME Foundation was instituted. The mandate of the Foundation is "to further the goal of the GNOME Project: to create a computing platform for use by the general public that is completely free software" (The GNOME Foundation 2000). The Foundation fulfills the following four roles (Mueth and Pennington 2002): (1) it provides a democratic process in which the entire GNOME development community can have a voice; (2) it is responsible for communicating information about GNOME to the media and corporations; (3) it will guarantee that the decisions on the future of GNOME are done in an open and transparent way; (4) it acts as a legal entity that can accept donations and make purchases to benefit GNOME.

The Foundation comprises four entities: its members (any contributor to the project can apply for membership); the board of directors (composed of 11 democratically elected contributors, with at most 4 with the same corporate affiliation); the advisory board (composed of companies and not-for-profit organizations); and the executive director. As defined by the Foundation's charter, the board of directors is the primary decision-making body of the GNOME Foundation. The members of the board are supposed to serve in a personal capacity and not as representatives of their employers. The Board meets regularly (usually every two weeks, via telephone call) to discuss the current issues and take decisions in behalf of the entire

community. The minutes of each meeting are then published in one of the GNOME mailing lists (foundation-announce).

Committees

Given the lack of a single organization driving the development according to its business goals, OSS projects tend to rely on volunteers to do most of the administrative tasks associated with that project. In GNOME, committees are created around tasks that the Foundation identifies as important. Contributors then volunteer to be members of these committees. Examples of committees are the GUADEC team (responsible for the organization of the GNOME conference), the Web team (responsible for keeping the Web site up to date), the sysadmin team (responsible for system administration of the GNOME machines), the release team (responsible for planning and releasing the official GNOME releases), the Foundation membership team (responsible for maintaining the membership list of the foundation), and several others.

The Release Team

In an OSS project that involves people from different organizations and with different time commitments to the project, it is not clear how to best organize and keep track of a release schedule. GNOME faced the same difficulty. Each individual module might have its own development timeline and objectives. Planning and coordination of the overall project is done by the release team. They are responsible for developing, in coordination with the module maintainers, release schedules for the different modules and the schedule of the overall project. They also keep track of the development of the project and its modules, making sure that everything stays within schedule. Jeff Waugh, a GNOME Foundation member, summarized the accomplishment of the team and the skills required in his message of candidacy to the Board of Directors in 2002:

> [The release team] has earned the trust of the GNOME developer community, madly hand-waved the GNOME 2.0 project back on track, and brought strong cooperation and "the love" back to the project after a short hiatus. It has required an interesting combination of skills, from cheerleading and Maciej-style police brutality to subtle diplomacy and "networking."

Case Study: Evolution

Ximian Evolution evolved within the GNOME project as its groupware suite based around a mail client. The project started in the beginning of

1998. At the end of 2000, Ximian (previously called Helixcode, after its creating company founded by Miguel de Icaza) started operations and decided to take over the development of Evolution. By the end of 2002, Evolution was composed of approximately 185,000 lines of code, written mostly in C. 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 FS product with functionality similar to Microsoft Outlook or Lotus Notes (Perazzoli 2001).

As with most of the GNOME modules, the development environment includes CVS (used for version control of the files of the project), Bugzilla (used to track defects), one mailing list for developers and one for users, and a collection of documentation pages hosted at the Ximian and GNOME developers' Web sites.

As is required by the GNOME guidelines, when a contributor commits a MR, he or she modifies the relevant changelog file to add a description of the current change. The commit triggers an e-mail message to the GNOME cvs-commits mailing list, which will include all the details of the transaction: who made it, when, the files modified, and the log message. Usually, the changelog and the CVS log messages indicate the nature of the change, list defect numbers from Bugzilla, and often include a URL pointing to the change and/or to the defect.

Figure 11.2 displays the number of MRs for each month of the project. Before Ximian was born, the level of activity was very low. It is also interesting how the number of MRs correlates to the number of releases, peaking just before version 1.0, at the same time that the frequency of small releases increased.

Figure 11.3 shows the net number lines of code added to the project (this number includes comments and empty lines) as a function of time. It also shows the number of new files added to the project (removed files are not taken into account) and correlates this information with the release dates. Some interesting observations can be made. There seems to be a correlation of added LOCS and new files, and the number of added LOCS and new files is flat in the month previous to release 1.0, suggesting a period in which debugging took precedence over new features. After release 1.0, the development was relatively flat compared to the previous period. From the evolutionary point of view, it is particularly interesting to see that during April 2002, more than 5,000 LOCS were removed from the project. Thanks to the changelog, it was possible to learn that the LOCS removed were automatically generated, and no longer needed. Being able to discover the reasons behind changes to a project emphasizes the

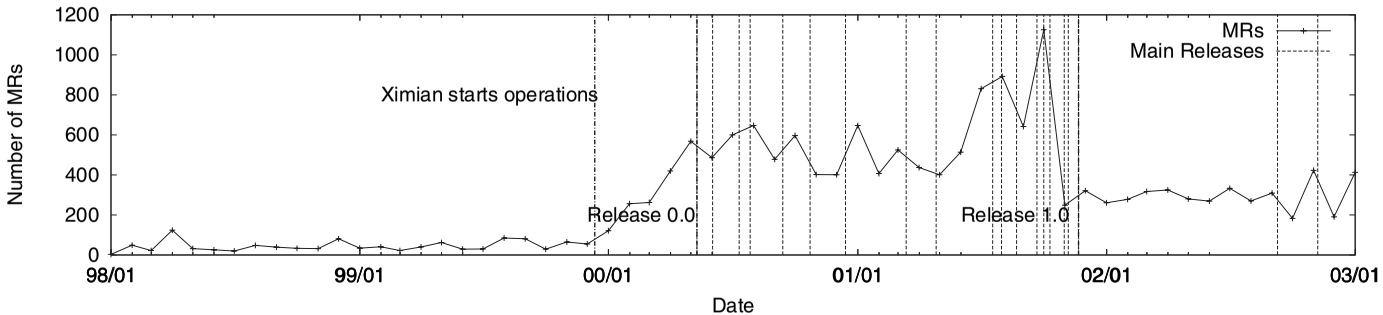

Figure 11.2
Number of MRs per month in Evolution. There is a significant increase in activity after Ximian starts operations, and the largest activity coincides with the release of version 1.0.

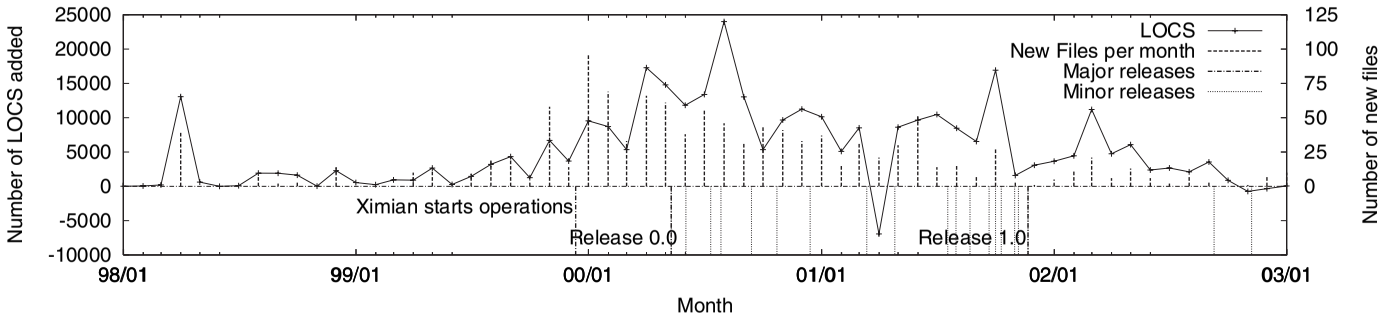

Figure 11.3
Growth of the source code of Evolution. The left axis shows the number of lines added to the project (sum of deltas equals lines added minus lines removed), while the right axis shows the number of files added each month.

importance of the changelogs in the development process. Readers are invited to read German and Mockus 2003 for a more in-depth analysis of Evolution.

Conclusions and Future Work

GNOME is a successful and mature FS project with a large number of contributors. It has been able to evolve from a purely volunteer effort into one that is mainly driven by the industry, while still allowing active participation by volunteers. Like many other OSS projects, a relatively small number of core contributors are responsible for most of the project. The source code is divided into modules, in which the interaction between contributors is minimized.

GNOME's development process is open. A large amount of data is available, tracing its history to the beginning. This data encompasses mailing lists, version control records, and bug tracking, giving the researcher the ability to inspect the code at a given moment in the past, and correlate it to its then-current defects and e-mail messages from its contributors. Some examples are shown on how this data can be used. Further and more in-depth empirical studies are needed to understand the interactions of its contributors, its architectural evolution, and its quality, for example. A comparative analysis of GNOME and KDE (GNOME's main competitor for the Unix desktop) will provide insight on how different (or similar) these two projects are in their development processes and architectures.

GNOME, like many other large OSS projects, provides a gold mine of data ready to be exploited. The resulting studies have the potential to provide a better understanding of how OSS evolves, identifying some good practices that could benefit many other OSS projects and, to a lesser extent, closed source projects too.

Acknowledgments and Disclaimer

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 Joe Feller, Scott Hissam, and Dan Hoffman for their invaluable comments during the preparation of this document. Any opinions, findings and conclusions expressed in this document are those of the author and do not necessarily reflect the views of the GNOME project or the GNOME Foundation.

Notes

1. The beginning of the project can be traced back to a license war between the advocates of FS and those using Qt, at that time a proprietary GUI library for X11, (a windowing system for the UNIX operating system) which was eventually released under the GPL and is the basis of the KDE project; it is similar in scope to GNOME. See Perens 1998 for a discussion of this issue.

2. Sometimes when some of the developers of an open source application disagree, they decide to create an alternative version of the project that continues its life independently from the original project. This new application is known as a *fork* of the original one.