The Contribution of Free Software to Software Evolution

Andreas Bauer, Markus Pizka
Institut für Informatik, Technische Universität München
Boltzmannstr. 3, Germany – 85748 Garching
{baueran|pizka}@in.tum.de

Abstract

It is remarkable to think that even without any interest in finding suitable methods and concepts that would allow complex software systems to evolve and remain manageable, the ever growing open source movement has silently managed to establish highly successful evolution techniques over the last two decades. These concepts represent best practices that could be applied equally to a number of today's most crucial problems concerning the evolution of complex commercial software systems. In this paper, the authors state and explain some of these principles from the perspective of experienced open source developers, and give the rationale as to why the highly dynamic "free software development process", as a whole, is entangled with constantly growing code bases and changing project sizes, and how it deals with these successfully.

1 Introduction

> "Virtue is more to be feared than vice, because its excesses are not subject to the regulation of conscience." — Adam Smith (1723–1790)

The development and continuous evolution of open source software exposes astonishing similarities with the ideas of free market economies [29]. The liberal use and manipulation of software practised in open source environments shares the principles of economic liberalism by establishing unrestricted trade, growth, and manipulation of software. Consequently, it could be argued that the evolution of products and processes in open source environments will prove to be superior to any other model in the long run, because the self-regulating interplay between demand and supply provides for constant selection and improvement.

Indeed, there is growing evidence for this hypothesis. The initially rather insignificant open source community has grown to an extensive free market for software artefacts with thousands of participants world wide and innumerable products. Due to the enormous power of this market, the quality of some of its products, e. g. GNU/Linux, has already become sufficiently high for broad commercial deployment. In addition to this, the size and diversity of this open market allows it to respond quickly to changing requirements — see, for example, new device drivers. Besides products, the development process within this market evolves constantly, too. The emergence of structured communication platforms, such as SourceForge [5], the introduction of roles, e. g. "maintainers", and the dissemination of elements of agile methods [9] clearly substantiate this supposition.

We do not want to give the impression that open source development could be the "silver bullet" [14] of software engineering because, obviously, open source has its shortcomings and troubles, too. However, we do argue that the long history of free software has introduced concepts which are potentially useful to help large software projects, consisting of millions of lines of code to evolve and develop in a healthy manner, even over decades. Our own hands-on experience with some of the high profile open source products such as the GNU Compiler Collection (GCC), or GNU/Linux, as well as our authorship and involvement with smaller open source products, made many benefits of the free software development model, as well as of the resulting architectures, evident to us.

We suggest that several of the principles and practices discussed below, can and should be adopted by non-open (commercial) software organisations to improve their evolution capabilities for both products and development processes.

Outline

In § 2 we clarify the scope and context of our work. It defines the term "open source" more precisely as *free software*, describes related work with reference to successfully evolving free software projects, and it puts this into context with descriptions of research on software evolution. After that, in § 3, we will take an in-depth look on the evolution of

the process in free software environments before we focus on the corresponding evolution of the technical architecture in § 4. In § 5 we conclude the paper by summarising the major results of our study, and by discussing their transferability to non-free, i. e. commercial, software systems.

2 Open source and free software

When we refer to open source software in this paper, we do not restrict ourselves to the "Open-Source" movement which was largely sparked by the GNU/Linux project [3]. We rather talk about free software in general and discuss software with open sources that are older than GNU/Linux and not, in particular, free as in "free speech", but rather free as in "free beer". The main difference being that, free speech is primarily regarded as a fundamental human right, while free beer is merely enjoyable, because it does not cost anything.

According to the Free Software Foundation's (FSF) GNU General Public License [15] — the license of the GNU/Linux project — source code may be modified and used by a third party as long as these program changes remain just as free and open and the original copyright notices are not altered. The FSF believes that this is the best way to accomplish the people's right to see and also alter a program's source code. The BSD-style licenses [36] on the other hand, are much more liberal and basically allow a third party to do whatever it pleases with the sources — even distribute modifications as "closed source".

The remainder of this paper is concerned with describing the authors' experiences with *free* software in general, rather than a particular occurrence of it.

2.1 Myths and clichées

It is a common misconception to think of "open source" as a purely chaotic process which results in a — more or less — usable software product, created by spare time hackers who do not share the same maintainability concerns as their "professional" or academic counterparts. Unfortunately, the title of Eric Raymond's seminal paper "The Cathedral and the Bazaar" [27] can be misunderstood as supporting the chaos clichée.

Of course, this is far from being true. As we will show below, free software development is often very well organised, with structured processes and well defined roles. Consequently, there is a large number of free software products which are considered to be at least equal in quality to their commercial counterparts and are, therefore, supporting our thesis.

Unfortunately though, the free software movement largely ignores the results and trends of academia (not only) in the field of evolution research and the other side ignores the circumstances under which their favourite text editors, compilers, or operating systems were brought to life. Again, we aim to narrow this gap between the achievements of free and "professional" software engineers in academia and industry alike.

2.2 Reality

To name just a few of the successful free software systems we would like to point out, in particular, the BSD-based operating systems FreeBSD, NetBSD, OpenBSD, and Darwin [2, 6], which originate from ideas and code created mostly in the 1980s. Thanks to the high quality of the products and the "free beer" philosophy of the BSD license (i. e. it simply does not cost anything), major vendors like Apple even based their commercial operating system on an open source kernel named after Charles Darwin, who gave birth to the theory of evolution [13, 19].

Even "free speech" projects like GCC [32], or GNU/Linux are driven in large parts by the financial involvement of major companies (e. g. IBM, Red Hat) who assign skilled developers and other monetary resources to software products where everyone can read and modify the source code [17]. The outcome, of course, must be a maintainable product in one way or another, because maintainability is a major reason why "dinosaur" projects like GCC, *BSD, Emacs, GNU/Linux, etc. still play an important role today, each in their individual fields.

Obviously, the open source movement has developed its own concepts and techniques that help big projects evolve successfully over time as well as under constantly changing requirements. Although many of the free software products are not interesting anymore in terms of technical innovation, they are all but irrelevant today. Hence, part of the evident success of (say) an operating system like GNU/Linux must be related to the way this software adapts to a changing technical reality community, as well as with increasing numbers of contributors to the project (see § 3.1, 3.2).

2.3 Related work on evolution

Due to the large gap between commercial software development, academic research, and the practices of the free software community it comes as no surprise that there is still little research directly targeted at the evolution of free software besides [23], [20], [33], and [16]. On the other hand, an extensive variety of evolution concepts and techniques, such as configuration management (e. g. CVS), regression testing [28], refactoring [25], source code analysis, code generators, and separation of concerns — to name only a few — play an important role in the free software world, too. We deliberately abstain from enumerating this extensive list, but it should become obvious that the evolution of

free software as it is described below has links to various concrete evolution techniques. However, in this work, our interest is focused on the *principles* of the evolution of free software, independent of certain techniques.

Lehman's well known eight laws of software evolution (from [21] to [22]) address the fundamental concepts underlying the dynamics of software evolution. As we will see, free software complies exceptionally well with Lehman's laws, although both the laws and the free software development process emerged independently: free software is continuously changed (law I), complexity increases noticeably (law II), and the self-regulation of the evolution process is obvious (law III). Our observations described in the following also strongly support laws IV to VIII. On the contrary, the substantial compliance of free software development with Lehman's laws is one way of explaining the success of open source software development.

3 Evolution of the development process

Unlike a lot of closed source software projects, free software tends to start out with hardly any administrative overhead. Early project phases to structure and coordinate the following development process, such as proper requirements engineering, usually play no significant role at all. However, it should be obvious that, as the project size grows, the administrative efforts can not remain constant. There are many prominent examples that second this conclusion [20, 23, 12].

In fact, the free software development process is highly dynamic, scaling with the underlying architecture as well as with the number and skills of people involved in a project. However, there is not *the* single free software development process per-se, but an evolution which is firmly entangled with the complexity of the resulting product itself.

3.1 Inevitable technical changes

Some of the dynamics in the free software development process are due to technical changes, rather than developers' decisions. For example, the widely used compiler suite GCC was originally created in the mid 1980s to be a fast and practical C compiler for 32-bit platforms that address 8-bit bytes and have several general purpose registers [31]. Nowadays, GCC supports more than 200 different platforms and bit ranges [26], as well as many more programming languages and its core consists of over 900,000 lines of code. And, while the GCC project has started out with simple email transfers between core developers (later, via Usenet as well), its current development process works fundamentally different today. The project has, obviously, not only changed its original aims, but also the number and types of people who contribute and even the way they contribute to the constantly evolving product. The accompanying manual for GCC 3.2.2 [32] states the names of 302 different contributors which, of course, is a strong contrast to 1984 when Richard Stallman himself created the first versions of a — back then — rather simple system compiler for potential GNU platforms.

These drastic changes were possible, mainly because of the technical conditions that have improved vastly since the creation of GCC, especially due to the expansion of the Internet and all the new transfer protocols that were spawned during that process, e. g. Hypertext Transfer Protocol (http). In particular, the GCC project of today benefits strongly from the following technical advances:

- automated mailing list management with access to searchable archives and web interfaces to coordinate and review the efforts of a world wide distributed network of developers,
- (public) CVS servers with web interfaces which keep track of different versions as well as of independent developments within the project
- a huge number of (http and ftp) mirror sites which increase the availability of the relevant data world wide,
- the introduction of and interest in new languages (e. g. Java, Haskell) and hardware platforms (e. g. ia-64 architecture),
- a modern and automated bug tracking system which is accessible via the world wide web,
- "compile farms" offering central access to multiple platforms at once, and usually sponsored by industry firms with substantial interest in open source products,
- a growing number of external projects which are built on top of GCC, but manage their own progress independently (e. g. Glasgow Haskell Compiler (GHC), Realtime Java implementation, Mercury compiler),

This is also the case for other big open source projects, because most of them evolve by deploying the same tools (CVS, RCS, BugZilla, etc.) and under similar circumstances or technical conditions. Further examples are *BSD, Mozilla, and the Linux operating system kernel. If we compare the rise in numbers of contributors of, e. g. FreeBSD to GCC we see similarities between the two projects: in 1995 when FreeBSD 2.0 was released it had 55 contributors (i. e. people with write access to the code repository); by the end of 2002, a total of 319 people were allowed to commit their changes to the code base [20].

3.2 Change of organisation

The real challenge in establishing and maintaining a successful development process for a free software product is to introduce technical changes according to the accompanying social changes when a project surpasses a "critical size". "The critical size" does not only depend on lines of code but also on program modules, number of contributors and, basically, any other metric that allows us to draw conclusions about the complexity of a software product.

Table 1. The size of open source products*

Lines of code	Project	Age
2,437,470	Linux kernel 2.4.2	1991
2,065,224	Mozilla	1998
1,837,608	XFree86 4.0.3	1987
984,076	GCC 2.96-20000731	1984
967,263	GDB / DejaGnu-20010316	mid. 1980s
690,983	Binutils 2.10.91.0.2	mid. 1980s
646,692	glibc 2.2.2	early 1990s
627,626	Emacs 20.7	1984

*Source: David A. Wheeler's *More Than a Gigabuck: Estimating GNU/Linux's Size* [38].

The nowadays prospering free software projects have all surpassed this critical size at least once, usually in lines of code and often in the number of contributors as well. As can be seen in Table 1, Linux version 2.4 consists of more than 2,400,000 lines of code and, in addition, its current ChangeLog files suggest at least an equal amount of active developers are involved as in FreeBSD.

These numbers hint to the fact that the management tasks in successful free software projects can not be solved by a single person (e. g. the original author of a project) alone. Once a critical size has been reached an executive board emerges one way or another and the board coordinates and controls the further evolution of the product. Consequently, a mature project like GCC is not driven forward by Richard Stallman anymore, but by a steering committee that consists of 12 professional software developers, partly paid by software vendors (e. g. Red Hat, IBM, Apple) to focus on the development of the free compiler suite (see Table 2). While the members are all experts in their fields, it is not expected from any single member to keep a full understanding of the entire 900,000 lines unity as such.

As can be seen in Table 2, free software products seem to cope well with an executive board of 10–20 members. Obviously, this is large enough to manage a complex project, but small enough to ensure development does not cease due to internal debates and politics. Interestingly enough, the listed projects are roughly equivalent in size and age, but hardly connected to each other and still find 10–20 core members a comfortable size to make the "right" decisions about a project.

Table 2. Executive boards in free software projects

Project	Board name	Members
FreeBSD	Core	15
GCC	Steering Committee	12
Debian	Leader & Technical Committee	8
Mozilla	Project Managers ("Drivers")	13
KDE	Core Group	20

It should be pointed out that this strategy also perfectly suits the results of extensive studies of the success and failure of commercial software projects, such as the well known CHAOS reports of the Standish Group [35, 34]. These studies show a project's tendency to fail, the larger it gets. Hence, it is highly recommended to keep a project considerably small, or it is very likely to fail. This is why agile methods, such as XP, deliberately refuse to carry out "huge" projects in a single step. Unfortunately, the dissemination of this fundamental lesson into commercial environments seems to be very slow. In contrast to this, self-regulation due to competition and selection within free software projects provides for a suitable size of the project. The project gets split continuously into smaller subsets as needed, with new maintainance tasks assigned to core team members according to their skill, experience, and current project responsibilities. In effect, the entire organisation scales effectively with the overall size of the product.

However, the way an executive board is nominated differs between the various free software projects. While Debian, for example, uses democratic elections, the KDE team is solely merit based. Anyone can become a member of the KDE core group, but the particular applicant must have distinguished him/herself through outstanding contributions and dedication in the KDE project over a considerable period of time. Albeit, KDE and Debian both make sure that the most suited and skilled people are in charge, rather than the "loudest", oldest, youngest, etc. as it often happens in commercial environments. This, in itself, is an evolutionary process that puts forth skill rather than pure authority.

3.3 Change management

Despite the similarities between the various "big" free software projects, there are many different approaches to handling change management. These are, however, converging more and more due to the deployment of similar

utilities (CVS, RCS, BitKeeper, patch/diff, etc.) that are automatising the, in parts, very complex tasks involved.

As Kumiyo Nakakoji et al. [23] have observed, the various approaches in open source change management seem to be equally successful as there are prominent examples for the individual techniques. In their paper, different histories of change management are called "Evolutionary Patterns" which distinguish themselves mainly by the way patches are tested, reviewed, and merged into a project.

Linus Torvalds and Alan Cox, for example, two driving forces within Linux kernel development, have raised concerns in several interviews about using public CVS servers to handle change management even though, internally, many important Linux modules are concurrently version controlled [30]. GCC and *BSD, on the other hand seem to be comfortable with the facilities and possibilities provided by a public CVS server, despite occasional "commit wars" [20]. We talk about a commit war when people working on the same parts of code keep overwriting each others changes. This can not happen with the main Linux kernel, because Linus Torvalds as the project's leader, decides personally which patches should be adopted and which should rather be dropped or picked up by other maintainers for their individual trees.

In terms of change management, open source projects seem to be doing the same things that can be found in closed source software (see also [37]). It should be pointed out though that even CVS as a tool has its roots in the free software world [11], which just shows how an open source program that started out as a couple of Shell scripts posted to Usenet evolved into a de-facto standard today — simply because it was always tightly integrated into the evolution of the other free software products it aimed to support. That is, the numerous products developed with CVS have, effectively, triggered the evolution process of this toolkit, up to the point where it became an irreplaceable cornerstone in both free and commercial projects.

4 Evolution of the architecture

In a free software project, it is not only the social structure which is adopting to a changing reality, but also the architecture itself that is very often subject to a natural evolution, rather than the result of (say) a carefully planned requirements analysis. Again, one good example supporting this claim is the GCC project's recent adoption of the Cross Vendor Application Binary Interface proposed by Intel which, initially, received a negative response from its users, because it introduced incompatibilities with previous C++ programs. The step was necessary though in order to be compatible with binaries created by other (commercial) compilers that also support code generation for the new Intel (64-bit) hardware as well as support for the majority of features proposed by the latest ISO standard for the C++ programming language [18].

4.1 Modular code and layers

Changing the ABI of an existing compiler is not a trivial task at all although, in the case of GCC, it was possible due to the modular design of the architecture that allows the majority of the code generation to happen in a platform-independent manner [31] (see Fig. 1). That is, many of the back end modifications are transparent to the final compilation passes where the RTL representation gets mapped onto machine language templates that describe specific platform properties.

Other projects like the Linux kernel, Mozilla, or *BSD have similar logical and physical program modules, enabling a more optimal coordination of the contributor's efforts and a better approach to change management. The majority of these modules however, were not necessarily obvious and thus present when these projects were initiated more than ten years ago. Therefore, the sensible differentiation must result from the evolution process that is happening as an increasing number of contributors submit new source code, ideas, and solutions to improve already existing code.

4.2 Entanglement of process and architecture

We argue that, in well maintained, successful free software projects, the technical structure of the underlying architecture is always entangled with the organisation of the project. In other words, every change to the organisation of a project must result in a change of the technical composition of the underlying architecture, and vice versa.

Fig. 2 shows the intense correlation of GCC's technical structure and the overall organisation of the project. It also shows how additions to the code base are being coordinated via a public review on mailing lists first, in order to eliminate obvious bugs before they are being committed to the CVS repository. In GCC, even the maintainers themselves go through this process to inform the rest of the contributors about the upcoming changes. In fact, it is not uncommon for a patch file to evolve during this process, even before it is an integral part of the GCC suite. The reason is that large or complex patches have to be well understood before they can be adopted and, therefore, a process of constant refinement is necessary to converge the new functionality with the ever changing code base. Our own experience in GCC development has shown that this process sometimes takes weeks, and even months [8].

In consequence, the organisation and reorganisation of the contributors' efforts is, as we claim, a "natural" process driven mainly by necessity and rationale, rather than by authority. Therefore, due to this strong entanglement of archi-

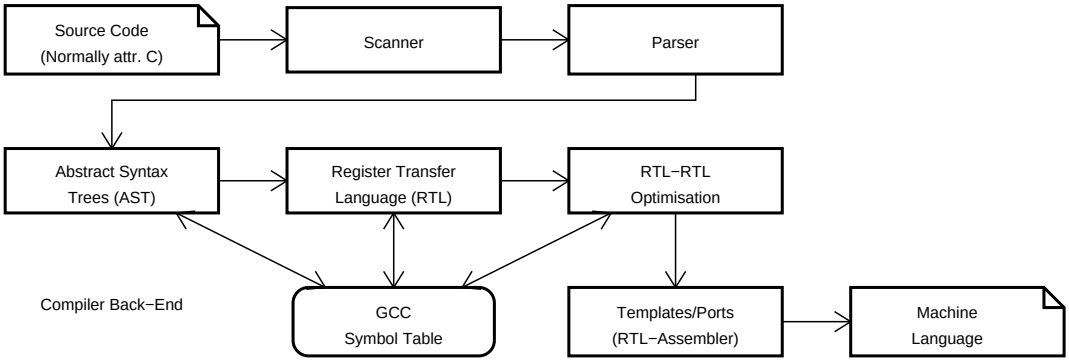

Figure 1. Logical and physical program modules of the GCC core

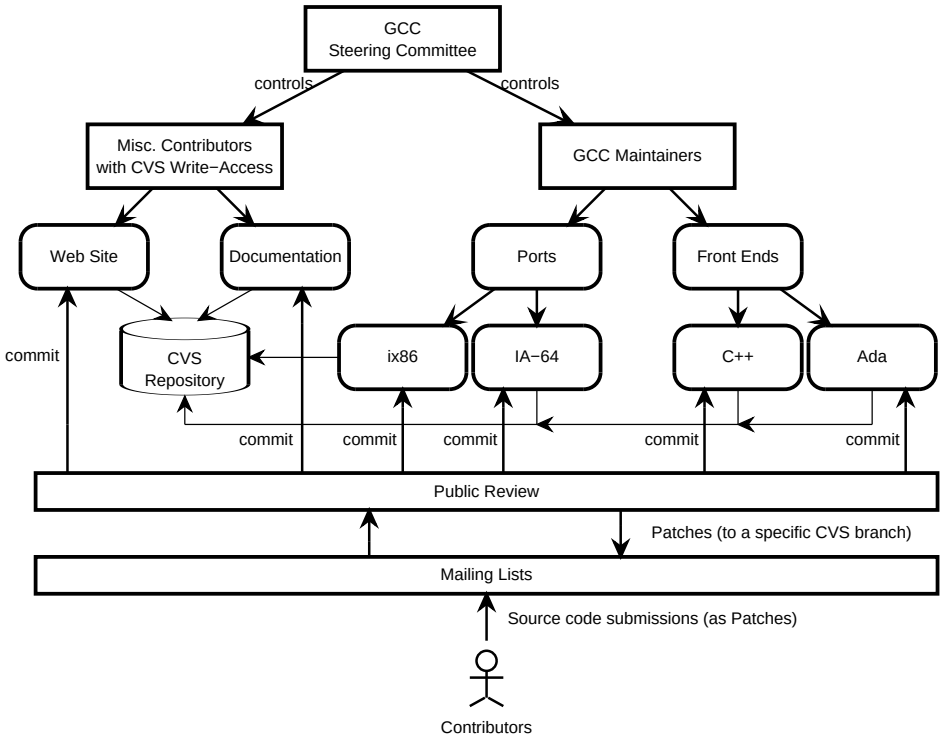

Figure 2. The main parts of GCC's project structure and organisation

tectural decisions and a project's organisation, a lot of the technical structure must be evolving "naturally" as well: the solution which is the most practical, robust or maintainable one succeeds (maybe not instantly, but certainly asymptotically).

4.3 Project and interproject dependencies

Even though commercial software vendors are concerned about the evolution of their software, they often have a static, product-oriented project organisation which is in strong contrast to the semantics of the word "evolution" as such. In fact, there are cases where the project structure in closed source projects reflects the geographic distribution of the company, rather than the purpose and goals of the product. For example, a team in city A is assigned task a, and a team in city B is responsible for task b. Is software really expected to evolve in a sensible manner under such static conditions?

Another popular open source project with commercial roots backs our theses. When Netscape released the "Communicator" sources in 1998 [12], it gave birth to the Mozilla project which, for many years, was an example of an unmaintainable, thus, unsuccessful open source project. Nowadays Mozilla is fundamentally different from Communicator though and the project has undergone many architectural changes that also led to successful commercial applications of newly emerging Mozilla components. Fig. 3 illustrates how the complexity of this once single large product has been systematically broken down into more manageable projects that, as splitting up continues, gain more flexibility on their own and are becoming more independent from the main Mozilla organisation as well.

As a matter of fact, in April 2003 the Mozilla project leaders released a statement to announce their biggest architectural split so far. Mozilla as such will only remain as the project's moniker while its core components, the mail, news, and web browsers are all turning into separate projects as it is reflected in the new roadmap: "The reasoning behind these new roadmap elements comes down to preferring quality over quantity. We must do less, but better, and with sound extension mechanisms [4]."

At the moment, Mozilla consists of approximately 50 "core projects", over two million lines of code, has 13 project leaders, and about 1000 active contributors [7]. The project has, finally, gotten rid of its old unmaintainable roots and is prospering, so much so that Netscape bases their browsers on Mozilla these days — and not vice versa.

4.4 Reduction of complexity by code splitting

A simple descriptive calculation helps one to comprehend the success of this separation strategy. From the results of cost estimation research, e. g. Function Points and COCOMO [10], the diseconomy of scales is well known, i. e. the cost of a software project grows non-linear with its size. Based on this experience, the COCOMO approach models the cost of a project with the formula $A \cdot n^B$, where n is the size of the source code in KLOC (1000 lines of code) and $A, B > 1$. Now, let us assume a piece of code consists of n modules (interchangeable with KLOC). We make the simplifying assumption that the program dependencies inside this piece of code cause quadratic growth of complexity, i. e. $B = 2$. Hence the overall complexity can be expressed as

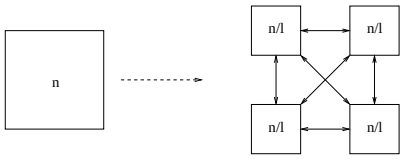

Figure 4. Code splitting

$$C_{old} = O(n^2) \tag{1}$$

If we divide the piece of code into l thoroughly separated parts, the overall system complexity decreases accordingly to

$$C_{new} = O\left(\frac{n^2}{l} + c \cdot l\right) \tag{2}$$

In other words, the complexity C_{new} of the whole product is significantly reduced. The absolute reduction is even more significant with growing n and l. The constant c is a linear factor and represents the new glue components that need to be introduced between the now separated l code pieces. Due to the isolation of the l parts, we can also assume that the new glue components do not cause quadratic complexity by themselves. Technically, this is achieved by using thin interfaces between the separated components, such as interprocess communication primitives provided by the operating system or, also, encapsulated data structures.

From an algorithmic viewpoint — not surprisingly — the complexity remains quadratic in (1) and (2), but in absolute numbers, C_{new} will always be smaller than C_{old} if n is sufficiently large compared to c. The maintainers of free software project provide for this sensible division of the code as soon as the envisaged evolution step and the comprehensibility of the code demand it.

4.5 Incremental rewrites

Successful free software projects share a strong ability to do incremental rewrites of hard-to-maintain source code. Mozilla is a very good example of this practice as today it

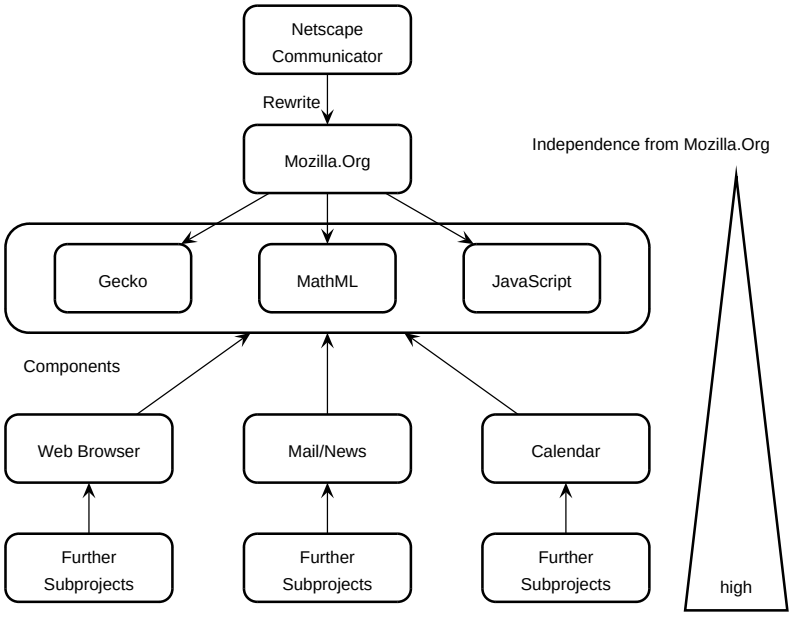

Figure 3. Interproject dependencies in Mozilla.Org

shares basically zero source code with its predecessor Communicator [7], although it was never rewritten as a whole in just one single step.

Netscape, in contrast, made the single worst strategic mistake any software company could make: they decided to wait for the complete rewrite of their code base, in this case done by the Mozilla project. While, during that time, Netscape was forced to skip version 5.0 of Communicator, they had lost valuable market share mainly to the Internet Explorer by Microsoft.

Albeit, for the Mozilla project being a free and open organisation without significant pressure from the market, it was a huge success and it even lead to industrial-strength components to be found in many commercial applications (e. g. Borland Kylix API, AOL web browser, Netscape Communicator, various embedded web browsers for mobile phones and PDAs).

This case shows the huge discrepancy between the outcome of an intention applied in a commercial environment, compared to the outcome of that same intention applied in a free environment. It succeeded in the free environment due to flexible, self-regulatory evolution, but failed in the other because of ill-lead and unrealistic planning.

If we consider the Linux kernel as a rewrite of AT&T UNIX, BSD, or MINIX, we can come to similar conclusions: a decade ago, it would have been a disaster for any company to make the immature Linux a building block of their business model, but for the free software community these last ten years of intense development have turned out to be extremely successful as we can clearly witness today. And even in the current Linux kernel, major incremental rewrites are still happening, with the latest infamous example being the (disputes over the) restructuring of the IDE layer (see kernel mailing list archives).

The GCC suite, being an even older open source project than Linux and Mozilla is subject to major changes as well. The AST project's goal is to rewrite the Abstract Syntax Tree handling in the back end, and to shift a lot of the Register Transfer Language optimisation into the tree optimisation passes [1]. There are several subprojects, such as the Single Static Assignment branch (SSA), mainly sponsored by Red Hat, to lead this distributed and incremental rewrite process to a success [24]. It should be obvious that, after 20 years of existence, it is no trivial task to restructure a compiler's back end, but previous, successful rewrites hint to the fact that an open source project's success is firmly connected to its ability to restructure itself when and where it becomes necessary. This is, we believe, another important evolutionary principle in long living free software projects.

5 Conclusion

The evidently successful free software development model is a formidable source of good software evolution principles and practices.

In contrast to most commercial software projects, continuous and unrestricted evolution is intrinsic to free software products. Usually, the only constant in a free software project is constant change. Considering Lehman's laws on software evolution [22] it is not surprising that this strategy

has produced long living and high quality software products.

The process of free software development is all but chaotic. Instead, the process and organisation scales with the size of the project. That is, projects start with almost zero overhead and are able to grow rapidly from the beginning. When a certain size is exceeded, however, regulations, steering committees, and tools are added as needed; i. e. the process evolves. The resulting organisation correlates strongly with the technical structure of the product and not with the geographic location of teams, or a company organisation chart.

Commercial software organisations could benefit strongly from process evolution according to the technical product, too. They often employ a single structured process that either has to be carried out rigidly for all projects, or must be tailored to projects first. There is, usually, no evolution of the process matching the needs of the project as in free software. In addition to this, commercial software projects often fall far short of assigning the right people to the right task.

Natural competition and selection within the free software process emphasises skill, rather than authority and rank. This increases the quality of the outcomes. It is very likely that some kind of competition, combined with dynamic assignment of roles within companies, would also increase the quality of non-free software products. Of course, this would require a radical cultural change within most organisations, but thanks to agile methods, some of these changes are already disseminating into the commercial environments.

Besides the evolution of the process, some of the most interesting principles to learn from arise through the close entanglement of a changing development process and the evolution of the technical architecture, i. e. the product itself. First, the architecture of the product is hardly preplanned, but it evolves freely with changing requirements as well as with the size of the product. At certain points in time, the architecture and organisation are split into rather isolated parts leading to independently maintainable modules, or even completely new products.

The evolution of the architecture is accompanied by incremental code rewrites. The scope of rewrites is determined solely by the required change and the available resources. In contrast to non-free environments, rewrites are not constrained by non-technical aspects, such as the lack of rights, or static responsibilities. This, in turn, reduces the necessity of expensive and inefficient "work-arounds" that add to the complexity and decrease quality and maintainability alike. A piece of free software evolves in a healthy and natural way by incremental rewrites and sensible additions of code.

The observations summarised so far support our initial thesis that, compared to non-free environments, the free software movement delivers superior evolution strategies, similar to economic liberalism that does so for business.

Acknowledgements: This work was sponsored by the German Federal Ministry for Education and Research (BMBF) as part of the project ViSEK (Virtual Software Engineering Competence Center).

References

[1] Abstract Syntax Tree Optimizations. Web site, `http://www.gnu.org/software/gcc/projects/ast-optimizer.html`, 2003.

[2] Darwin — Open Source. Web site, `http://http://developer.apple.com/darwin//`, 2003.

[3] Linux.Org. Web site, `http://www.linux.org/`, 2003.

[4] Mozilla Development Roadmap. Web site, `http://www.mozilla.org/roadmap.html`, 2003.

[5] Open source development network. Web site, `http://sourceforge.net/`, 2003.

[6] OpenDarwin. Web site, `http://www.opendarwin.org/`, 2003.

[7] The Mozilla Project. Web site, `http://www.mozilla.org/`, 2003.

[8] A. Bauer. Compilation of Functional Programming Languages using GCC — Tail Calls. Master's thesis, Department of Informatics, Technische Universität München, Munich, Germany, 2003.

[9] K. Beck. Embracing change with Extreme Programming. *Computer*, 32:70–77, Oct. 1999.

[10] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[11] P. Cederqvist et al. Version Management with CVS. Technical Manual, Signum Support AB, 1993.

[12] D. Cubranic and K. S. Booth. Coordinating open-source software development. In *Eighth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 61–65, Stanford, CA, USA, 16–18 1999. IEEE Computer Society Press.

[13] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. 1859.

[14] F. P. Brooks jr. *The Mythical Man-Month*. Addison Wesley, 1995.

[15] Free Software Foundation. GNU General Public License. `http://www.fsf.org/licenses/gpl.html`, Free Software Foundation, Inc., Cambridge, Massachusetts, 1991.

[16] M. W. Godfrey and Q. Tu. Evolution in Open Source Software: A Case Study. In *Proceedings of the ICSM 2000*, pages 131–142, San Jose, CA, 2000.

[17] N. Harris et al. *Linux Handbook / A guide to IBM Linux solutions and resources*. IBM International Technical Support Organization, 2002.

[18] Intel Corporation. *Intel Itanium Software Conventions and Runtime Architecture Guide*. Intel Corporation, Santa Clara, California, Intel document SC–2791, Rev. No. 2.4E edition, 2001.

[19] H. Kingman. BSD: Darwin's theory of E-volution. *ZDNet Australia Tech News*, 2000.

[20] G. Lehey. Evolution of a free software project. In *Proceedings of the Australian Unix User's Group Annual Conference*, pages 11–21, Melbourne, Australia, Sept. 2002.

[21] M. Lehman. The programming process. Technical Report RC2722, IBM Research Centre, Yorktown Heights, NY, Sept. 1969.

[22] M. Lehman and J. F. Ramil. Rules and tools for software evolution planning and management. *Annals of Software Engineering*, 2001.

[23] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the international workshop on Principles of software evolution*, pages 76–85, 2002.

[24] D. Novillo. Tree SSA — A New High-Level Optimization Framework for the GNU Compiler Collection. In *Proceedings of the Nord/USENIX Users Conference*, Västerås, February 2003.

[25] W. F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, Urbana-Champaign, IL, USA, 1992.

[26] M. Pizka. Design and Implementation of the GNU INSEL Compiler gic. Technical Report TUM–I 9713, Technische Universität München, Munich, Germany, 1997.

[27] E. S. Raymond. The cathedral and the bazaar, 11 Nov. 1998.

[28] R. Savoye. Dejagnu – the gnu testing framework. Technical report, Free Software Foundation, 2002.

[29] A. Smith. *The Wealth of Nations*. 1776.

[30] J. Southern and C. Murphy. Eine zielgerichtete Explosion. *Linux Magazin*, (1), 2002.

[31] R. M. Stallman. GNU Compiler Collection Internals. `http://gcc.gnu.org/onlinedocs/gccint/`, Free Software Foundation, Inc., Cambridge, Massachusetts, 2003.

[32] R. M. Stallman. Using the GNU Compiler Collection. `http://gcc.gnu.org/onlinedocs/gcc-3.2.2/gcc/`, Free Software Foundation, Inc., Cambridge, Massachusetts, 2003.

[33] G. Succi and A. Eberlein. Preliminary Results from an Empirical Study on the Growth of Open Source and Commercial Software Products. In *Proceedings of the Workshop on Economics-driven Software Engineering Research, EDSER 3*, pages 14–15, Toronto, Canada, 2001.

[34] I. The Standish Group International. Chaos, 1995.

[35] I. The Standish Group International. Chaos: A recipe for success, 1999.

[36] University of California, Berkeley. The BSD License. `http://www.opensource.org/licenses/bsd-license.html`, Regents of the University of California, 1998.

[37] A. van der Hoek. Configuration management and open source projects. In *Proceedings of the 3rd International Workshop on Software Engineering over the Internet*, Limerick, Ireland, June 2000.

[38] D. A. Wheeler. More Than a Gigabuck: Estimating GNU/Linux's Size. Web site, `http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html`, 2002.