Coordinating Open-Source Software Development

Submission for 7th Workshop on Coordinating Distributed Software Projects

Davor Čubranić
Department of Computer Science
University of British Columbia
cubranic@cs.ubc.ca

1 Introduction

In the recent years a form of software development that was previously dismissed as too ad-hoc and chaotic for serious projects has suddenly taken the front stage. With products such as Apache, Linux, Perl, and others, open-source software has emerged as a viable alternative to traditional approaches to software development. With its globally distributed developer force and extremely rapid code evolution, open source is arguably the extreme in "virtual software projects" [1], and exemplifies many of the advantages and challenges of distributed software development.

According to its (trademarked) definition, open-source software (OSS) is software for which the source code is distributed or accessible via the Internet without charge or limitations on modifications and future distribution by third parties [10]. While much of the early ARPANet and Unix software was distributed in this manner, during the 1980's more ambitious open-source projects such as Free Software Foundation's GNU were started and gained support of developers across the Internet. However, it wasn't until the 1990's that open-source software development truly gained momentum and become synonymous with highly distributed development characterized by frequent iterations, thanks to the wide availability of the source code and openess to contributions from the community.

Today, open-source software dominates the Internet infrastructure — for example, in February 1999, over 54% of Web servers ran Apache server software [17], while it is estimated that Sendmail now handles about 80% of Internet e-mail [4] — and established computer companies such as IBM and Apple are starting to include OSS into their products [9, 2].

The main reason for this success is the growth of the Internet, which made collaboration between programmers feasible on a scale much larger than was possible before. With the global computer network in place, a huge pool of potential developers and testers willing to put their time into projects they found interesting or useful became available. Not surprisingly, this openess and fluidity also put unique demads on the development process. To cope with those issues, open-source software projects evolved their own methods and organization. This methodology has the potential to alter the whole approach to making software — resulting, its proponents would say, in more reliable products and faster and leaner development. However, it also faces some significant obstacles if it is to continue successfully growing. In my opinion, these obstacles cannot be surmounted by simply attempting to transplant ideas from the more traditional development models, even those that are also Internet-based. What is needed instead is research that will examine open-source software and development process, in light of their current accomplishments and respecting their specificities, and find what works, what doesn't, and how it can be further improved.

In this position paper, I will look at the ways some of the major and most successfull open-source projects deal with the issue of coordination among their many contributors. Although each of the projects examined here developed some unique practices, there are also significant commonalities. I will then indicate some of the problems caused by the existing practices, and put forward some possible approaches to OSS coordination that could make open-source software development more efficient.

2 Current Practices

There are hundreds, if not thousands, of open-source projects that are currently under development. The ones I include in this section are just some that are notable for their influence, size, and success, and which are consequently representative of many open-source development practices.

2.1 Representative Open-Source Projects

Linux is arguably the most well-know open-source project today. It is a Unix-type operating system kernel which aims for a complete implementation of the POSIX specification, with System V and BSD extensions. What started off in 1991 as a hobby project of Linus Torvalds, then a student at University of Helsinki, has evolved into a full-featured modern OS (consisting of more than 1.5 million lines of code) that accounted for 17.2% of server operating systems in 1998, an increase of 212% over the previous year [23], and has a user community that numbered over 7 million in early 1998 [25, 16]. Today, Torvalds continues to lead the project, which has become the open-source's poster child.

The Apache Web server originated in the early 1995 as a set of patches to the then-popular HTTP server from NCSA (hence the name, "A PAtCHy server"). These patches were collected by a group of volunteers from contributions from Webmasters frustrated by NCSA's lack of further development and then released back to the Web community. The patches were a big success, and soon the group moved on to a complete overhaul and redesign of the server: Apache 1.0 was released to the general public on 1 December 1995 and went on its march to the Web server market domination — according to Netcraft's February 1999 survey, 54% of Web servers run Apache [17]. The initial volunteers have then formed the Apache Group, which continues the development of the project.

InterNetNews (INN) is a complete Usenet news system, providing a set of servers for both news relaying and serving client newsreaders. It was written by Rich Salz [21] in the early 1990s, but as other obligations drew his attention away from further development, users took it upon themselves to share patches and generate unofficial releases. Eventually, Salz turned INN over to the Internet Software Consortium, a nonprofit corporation dedicated to production-quality reference implementations of the key Internet standards, which merged in the various unofficial releases and, prompted by the user community, placed the development under an open-source model.

Mozilla Web browser is probably one of the most interesting experiments in open-source software development, and could potentially determine the future attitude of commercial developers toward the open-source model. When in March 1998, Netscape released the source code for the next version of its Communicator Web browser under an open-source style license [18], it formed "the Mozilla Organization" to coordinate the developers' effort and act as a central point of contact for those interested in participating in the project. The Mozilla Organization (also known as mozilla.org) provides support for the developers (such as a Web site and mailing lists) and publishes Mozilla browser as its integrated version of the project's effort, although there is nothing to prevent others from creating and distributing their own versions. Netscape remains actively involved in the project: it provides funding and personnel for mozilla.org, and its employees have contributed some of the major subsystems. Also, code written by Netscape (and all its subsequent modifications) is covered by a slightly different license that gives Netscape additional rights, for example to reuse it in non-open-source projects like its Web servers. Undoubtedly, the success of this project would prompt other commercial developers to follow in Netscape's footsteps; its failure, on the other hand, could seriously undermine open-source's credibility in the software industry.

Perl is a general-purpose programming language, invented in 1987 by Larry Wall as a quick hack to simplify generating reports from systems logs [24]. It has since become the language of choice for small-to-medium projects, especially in the areas of World Wide Web development, system administration, text processing, etc. Today, the infrastructure for communication and coordination in support of the Perl community is provided by the Perl Institute, a non-profit organization "dedicated to keeping Perl available, usable, and free for all" [11].

2.2 Communication and Coordination

One of the most important characteristics of distributed software development is that developers cannot any more rely on face-to-face meetings, but have to make use of technology to allow them to communicate over distance. Researchers suggest that for effective distributed collaboration, developers of such projects need a rich array of tools offering both synchronous (collaborative editors, chat, and online meetings) and asynchronous capabilities [14]. However, open-source projects still, and almost without exception, primarily rely on mailing lists, sometimes gatewayed to newsgroups, for almost all communication activities. Furthermore, coordination among the participating developers in the form of discussions about the direction of the project, code review — for many projects even bug reporting and code contributions — is all conducted through such lists.

There are several reasons for choosing such a relatively low-tech approach. Firstly, email is essentially the lowest common denominator for Internet communication, which significantly lowers the bar for would-be contributors to start participating, or even just listening, in a project's development discussions. Secondly, the extremely distributed nature of even a core group of developers of OSS projects — for example, Apache's 20 core developers are located in five different countries across three continents — pretty much precludes the usage of synchronous communication. Thirdly, open-source development is extremely fluid, where structure is minimal and developers' contributions vary with time depending on their interest and other commitments. In such circumstances, it would be very difficult to impose a prescriptive coordination technology, such as for example workflow systems. Thus, most projects prefer to continue using email even after they have grown beyond the initial small group of developers, because it lets humans resolve any unexpected situations that may arise.

McGrath has noted that developing "social contracts" among participants in computer-mediated communication is often more effective than looking for a technological solution [15]. Given such a general-purpose medium such as email, it is not surprising that some projects have developed conventions specifically for coordination. For example, the Apache Project's active code repositories contain a file called "STATUS" which is used to keep track of the agenda and plans for work within that repository. The STATUS file includes information about release plans, a summary of code changes committed since the last release,a list of proposed changes that are under discussion, brief notes about items that individual developers are working on or want discussion about, and anything else that might be useful to help the group track progress. The active STATUS files are automatically posted to the developers' mailing list three times each week.

2.3 Version and Conguration Management

By the very definition of open-source, such projects have to have some sort of code repository on the Internet, where developers and other interested parties can access it for download. In the past, this was done mainly through anonymous FTP access to the recent versions (often there are at least the current "development" and "stable" versions). Also, typically the changes from the previous version are available as *patch files*, and are usually posted on the developers' mailing list for those who want to minimize the amount of necessary downloading while keeping current with the ongoing development. In the recent year or so, however, many (especially larger) projects have turned to CVS [3] to manage the code repository and ease the burden of version control and merging in individual submissions. Linux is a significant holdout in this department, because the "official" version is still put together and posted on the FTP site exclusively by Linus Torvalds.

The organization of configuration management in open-source projects stems from their internal developers' "hierarchy". Typically, there are two tiers of developers participating in the effort: a core group that is relatively small (for example, around 20 people in the Apache project), and a much larger pool of contributors. The core developers are actively involved (often on a daily basis) in the development of the product, and some minimum amount of commitment in terms of time and effort is usually implicitly expected. Contributors might submit an occasional bug fix or feature enhancement, as they have time, interest, or ideas. The core developers are then there to receive those contributions, review them, and integrate the accepted ones into the code base. Over time, contributors who have distinguished themselves by the quality and frequency of their work may be invited to join the core group and gain more responsibility in the project. In other words, open-source projects operate as *meritocracies* [19].

There are individual project differences, however: in Linux, the final authority and say on what goes into the kernel rests with Linus Torvalds, although the responsibility over subsytems has been delegated to the so-called "module maintainters" (who have often also written a major portion of the code they oversee). In Apache, the core developers form the Apache Group, which maintains control over the project, without further breakdown of the hierarchy by subsystem — essentially, all the members of the group have equal vote if an issue turns contentious. INN's development is also guided by a small core group (seven members at the time of this writing), while Mozilla's is divided by subsystems, with individual or small groups of "module owners" coordinating the development in their area — although, in theory, they are only the caretakes who could be replaced if the developers' community is dissatisfied with their work.

This sort of developer organization is reflected in the setup of the code repositories: in general the CVS source code repository allows read-only access to anyone on the Internet, while only the core developers have the permissions to directly modify the tree. Those programmers are responsible for evaluating and approving changes submitted by the community and integrating them into the tree.

3 Experiences and Criticisms

Perhaps because of its roots in the hackers' culture, although open-source development has been around for several decades, it hasn't gotten much attention from the software engineering community. Even descriptions of the open-source development process are still quite rare and largely anecdotal. In this section, I will summarize experiences with tools used for coordination in open-source projects and concerns about their capability to support further growth in complexity and features.

3.1 Tools and Coordination

In general, as Fielding and Kaiser noted in their description of the Apache project [6], the mailing list and its archives, CVS, and bug tracking systems (like Apache's GNATS, Linux kernel's Jitterbug, or Mozilla's Bugzila) support low-level coordination: "retaining project history, tracking problems and revisions, providing and controlling remote access, and preventing change collisions (the 'lost update problem')" (p. 89). However, a year and a half after their article, there is still no support for higher-level coordination — group decision-making, knowledge management, task scheduling and progress tracking, etc. — for any of the projects examined here. Design documents and development plans are at best jotted down in TODO-style files, often scattered around the source tree or the project's Web site, without an effective way to track progress of development efforts or even tasks that individual developers are currently engaged in. Even in Apache's case, with its regular posting of project's status and plans on the developer's mailing list, it is hard to gauge progress or sense the direction of evolution, since older version of the status messages are buried among hundreds of other messages in the mailing list archive.

Furthermore, the use of the mailing list as a primary communication channel among the developers has often resulted in a deluge of e-mail. For example, Linux kernel mailing list averages over 100 messages per day. As a matter of fact, if we define project coordination as "the attempt to get the **right information** to the **right people** at the **right time**" [8], it could be argued that open-source projects lack it entirely and leave all coordination work to humans. Although the mailing lists are for most projects automatically archived and available on-line for hypertext browsing, effective retrieval and management of that volume of information requires more sophisticated tools than basic reply-threading. This problem is even more acute for a developer who would like to join the project, since those archives often provide the only source of information on design choices and evolution of the system. This entry barrier is a problem that has been acknowledged by open-source developers [5, 7] and is all the more troubling because of the reliance of OSS projects on volunteer submission and "fresh blood".

All this points out that typical large open-source projects currently require an inordinate amount of manual and mental effort from their developers, who often have to rely on their memory and ad-hoc methods to compensate for the lack of adequate tools and automation. That the current OSS projects did so well under such circumstances is a testament to the talent of their programmers [5], although some are already questioning whether that talent will be in short supply if the open-source methodology becomes more widespread [13].

3.2 Managing Growth

Ted Lewis in his "Binary Critic" column in IEEE Computer [13] voiced a rare criticism aimed at the ability of open-source development process to cope with its own success. A big problem that open-source projects are going to face as they mature, Lewis claims, is that they will also grow in features and complexity, all the more so because of the "feature creep" brought on by ongoing user contributions. This will eventually overwhelm the resources of their handful of core developers and the capability of their typically ad-hoc organizational structure to cope with those stresses. Similar objections were also raised by Microsoft in an internal memo (subsequently leaked to the press): "The biggest roadblock for open source software projects is dealing with exponential growth and management costs as a project is scaled up in terms of innovation and size" [20].

While Lewis's article has generated a barrage of criticism on open-source-related mailing lists (which, because of their turn-around time, hasn't yet reached printed publications), and has been show to contain some serious factual errors, it also raises an important concern. True, Linux, for example, has managed to grow to over 1,500,000 million lines of code (comparable to other Unixes) over the last seven and a half years without compromises in its reliability while, at the very least, keeping pace with commercial operating systems. But it is also true that releasing version 2.2 took longer than expected and that occasionally there is a backlog of submitted patches awaiting Torvalds's approval for inclusion into the kernel. Mozilla, after a year of development and some significant progress, still doesn't have even an official public beta release, much less a production-quality version. These two are probably among the largest open-source projects in terms of the sheer number of lines of code, which may suggest that indeed we are reaching the point when the current coordination infrastructure is becoming insufficient and better tools are needed to assist the developers in more efficient collaboration.

4 Conclusion

As David Lawrence, one of INN's core developers, noted, open-source development model might not be the right choice for every software project, but it is still suitable for a wide variety of systems [12]. It would certainly be hard to argue against its effectiveness in the case of Linux, Apache, or Perl. In any case, as the software companies struggle to find a way to meet the often conflicting requirements of developing software faster while making it more robust, they might be forced to embrace alternative development models. Open source offers a development model that "creates a faster, leaner, and more reliable product than mainstream competitors, and does so essentially for free" [22],(p. 89). It is no wonder then that open-source software has recently been getting so much attention from the software industry and news media.

However, regardless of whether that attention proves to be only a fad, and the open source doesn't gain mainstream acceptance, it is unlikely that it will disappear. It has been around for over twenty years already and is only gaining in popularity; there are now even Web sites devoted to announcements and news related to ongoing and starting open-source projects (e.g., Freshmeat.net).

Based on published experience reports of open-source developers, more effective support for knowledge management and coordination will be needed as projects evolve and grow with time. The road awaiting for software engineering researchers is to gain better understanding of the dynamics and requirements of open-source development, which is necessary if the tools appropriate to open-source specificities are going to be developed. For example, enacting a traditional software process in an open-source project would not only be difficult, but could also be counter-productive because the potential contributors would simply give up if they perceive that they are giving away their freedom. What is needed are tools (and techniques, including social processes) that are flexible, complement the tools currently in use, and support the existing open-source development model almost transparently.

References

[1] K. Alho and R. Sulonen. Supporting virtual software projects on the Web. In *Proceedings of 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98) Workshop on Coordinating Distributed Sofware Development Projects*. IEEE Press, Apr. 1998.

[2] Apple, Inc. Apple announces Mac OS X server: New server software optimized for Internet, 5 Jan. 1999. Press release available at http://www.apple.com/pr/library/1999/jan/05osxserver.html.

[3] B. Berliner. CVS II: Parallelizing software development. In USENIX Association, editor, *Proceedings of the Winter 1990 USENIX Conference, January 22–26, 1990, Washington, DC, USA*, pages 341–352, Berkeley, CA, USA, Jan. 1990. USENIX.

[4] J. Edwards. The changing face of freeware. *IEEE Computer*, 31(10), Oct. 1998.

[5] R. T. Fielding. The Apache Group: A case study of Internet collaboration and virtual communities, 1997. http://www.ics.uci.edu/ fielding/talks/ssapache/overview.htm.

[6] R. T. Fielding and G. Kaiser. The Apache HTTP server project. *IEEE Internet Computing*, 1(4):88–90, July/Aug. 1997.

[7] F. Hecker. Mozilla at one: A look back and ahead, 2 Apr. 1999. http://www.mozilla.org/mozilla-at-one.html.

[8] H. Holz, S. Goldmann, and F. Maurer. Working group report on coordinating distributed software development. In *Proceedings of 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '98)*. IEEE Press, Apr. 1998.

[9] IBM Corp. IBM helps companies turn simple Web sites into powerful e-business solutions, 22 June 1998. Press release available at http://www.ibm.com/News/1998/06/223.phtml.

[10] O. S. Initiative. The Open Source definition, 1997. http://www.opensource.org/osd.html.

[11] T. P. Institute. http://www.perl.org/.

[12] D. C. Lawrence. InterNetNews server: Inside and open-source project. *IEEE Internet Computing*, pages 49–52, Sept./Oct. 1998.

[13] T. Lewis. The open source acid test. *IEEE Computer*, 32(2):125–128, Feb. 1999.

[14] F. Maurer and B. Dellen. An Internet based software process management environment. In *ICSE 98 Workshop on Software Engineering over the Internet*. Available online from http://sern.cpsc.ucalgary.ca/ maurer/ICSE98WS/Submissions/Maurer/ICSE.html, 1998.

[15] J. E. McGrath. Time matters in groups. In *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*, pages 23–78. Lawrence Erlbaum, Hillsday, NJ, 1990.

[16] J. McHugh. For the love of hacking. *Forbes*, pages 94–100, 10 Aug. 1998.

[17] Netcraft, Inc. Netcraft Web server survey, Jan. 1999.

[18] Netscape, Inc. Netscape announces plans to make next-generation Communicator source code available free on the Net, 22 Jan. 1998.

[19] T. A. Project. Apache project guidelines, 3 May 1998. http://dev.apache.org/guidelines.html.

[20] E. S. Raymond. Open source software — a (new?) development methodology, 1998. Annotated version of an internal Microsoft report available at http://www.opensource.org/halloween1.html.

[21] R. Salz. InterNetNews: Usenet transport for Internet sites. In USENIX Association, editor, *Proceedings of the Summer 1992 USENIX Conference: June 8–12, 1992, San Antonio, Texas, USA*, pages 93–98, Berkeley, CA, USA, Summer 1992. USENIX.

[22] J. Sanders. Linux, open source, and software's future. *IEEE Software*, 15(5):88–91, Sept./Aug. 1998.

[23] S. Shankland. Linux shipments up 212 percent. *CNET News.com*, 16 Dec. 1998. http://www.news.com/News/Item/0,4,30027,00.html?st.ne.ni.rel.

[24] L. Wall and R. L. Schwartz. *Programming perl*. O'Reilly & Associates, 1990.

[25] R. F. Young. Sizing the Linux market, second edition. White paper, Red Hat Software, 5 Mar. 1998. http://www.redhat.com/redhat/linuxmarket.html.