How the Cathedral Embraced the Bazaar, and the Bazaar Became a Cathedral

Terhi Kilamo[1], Valentina Lenarduzzi[2], Tuukka Ahoniemi[3], Ari Jaaksi[1], Jurka Rahikkala[4], and Tommi Mikkonen[5]

[1] Tampere University, Tampere, Finland; terhi.kilamo@tuni.fi, ari@linux.com
[2] LUT University, Lahti, Finland; valentina.lenarduzzi@lut.fi
[3] Tuxera, Espoo, Finland; tuukka.ahoniemi@iki.fi
[4] Vaadin, Turku, Finland; jurka.rahikkala@vaadin.com
[5] University of Helsinki, Finland; tommi.mikkonen@helsinki,fi

Abstract. Over the past 20 years, open source has become a widely adopted approach to develop software. Code repositories provide software to power cars, phones, and other things that are considered proprietary. In parallel, proprietary development has evolved from rigid, centralized waterfall approaches to agile, iterative development. In this paper, we share our experiences regarding this co-evolution of open and closed source from the viewpoints of tools, practices, and organizing the development work, concluding that today's bazaars and cathedrals have much more common characteristics than those that separate them.

Key words: Open source, development tools, software business

1 Introduction

In 1997, Eric S. Raymond [1] juxtaposed two ways of software development: the cathedral and the bazaar. The differences of the two have from the beginning been in the development approach instead of the source code alone. The cathedral model develops the software within a closed group of developers, and the product is only released in structured intervals; in contrast, the bazaar keeps the development process fast and visible to any interested party all the time.

Originally, the bazaar marked open source, and the cathedral proprietary. Over time, things have changed however, making proprietary development resemble open source software and its development models. In parallel, open source has grown to the level of importance where it has adopted many of the centralized development schemes traditionally associated with proprietary software. Moreover, open source quality increased year after year [2, 3, 4], and producers started to apply similar marketing models of proprietary [5]. As a result, companies started to consider open source as trustworthy of proprietary ones [6], and opening proprietary software has become a viable option [7, 8, 9].

We approach this evolution with a retrospective view of the past 20 years of software development, and find that the cathedral has turned into "a babbling bazaar" while the bazaar has evolved towards "reverent cathedral-building" [1].

2 Background

The origins of open source software can be traced back to 1974, when US Commission on New Technological Uses of Copyrighted Works (CONTU) [10] decided that software is a proper subject matter of copyright to the extent that it embodies its authors' original creation [11]. While previously code had been liberally shared among different stakeholders largely ignoring IPR, this was no longer an option, concerning in particular modifying program source code. This initiated the movement to regain the original 'freedoms' of programmers, eventually resulting in forming the Free Software Foundation (https://www.fsf.org).

The early years of open source software revolved around infrastructure software, including operating systems, windowing systems, programming tools and compilers, and other software that is commonly used to operate or program any computer. A key milestone in this era is year 1989, when the first version of the GNU General Public License (GPL) was published. It governs the use of open source software, taking open source to the league of a "serious" software system, where formal governance and ruling existed.

By year 2000, as witnessed by Raymond's essay [1], the focus was shifting to development models where proprietary software was compared to carefully crafting a pre-designed cathedral and open source was evolving organically based on individuals' desires in bazaars. In parallel, using open source as a part of business and proprietary software gained widespread interest [12]. Soon, open source become a viable business model [13], and it was adopted by various companies.

Today, open source is characterized by three key elements: (i) code comprising the implementation, (ii) the licence under which the software is distributed, and (iii) the community that maintains the code. Furthermore, open source is viewed through its meritocratic nature of collaboration where anyone can contribute, contributions are judged based on merit and the development is self-organizing [14]. This characterization does not consider the cathedral and the bazaar metaphor, and over time, the difference between the two has become a fine line – companies extensively participate in open source development [15] and open source communities and projects are organized and governed [16, 17]. The beginning of this transformation was documented by Fitzgerald [18]; the trend has since continued, making cathedrals embrace bazaars and bazaars to become cathedrals.

3 Why the Cathedral Embraced the Bazaar?

Pre-open source software industry habits were very similar to the rest of the industry, consisting of large enterprises that had very established ways and elaborated processes, to the brink of a disaster in their inability to adopt new practices [19]. For software development companies, this meant rigorous inspections [20] to improve quality and process models such as CMM [21], where the organizational capabilities were being improved without paying much attention to the interests of individuals.

Open source development challenged all the above. Since its humble origins, the power of open source has stemmed from programmers who choose to contribute to projects of their interest, not by management control. To support voluntary cooperation across the Globe, various tools were introduced, including source control management tools such as Subversion and later Git. Distributed version control systems have enabled the development through pulling code from repositories to create software locally as a combination of several code sources. The range of components from operating and windowing systems to tiny, almost self-evident snippets have made open source systems an integral part of systems [22], many of which we do not consider as open source, and these are also visible in software architecture of many systems [23].

Open source also helped to shape new development processes, such as introducing the pull requests mechanism and the concepts of continuous integration, deployment and delivery, as means for handling new contributions to the code base. Similarly, issue tracking tools such as Bugzilla and review tools such as Gerrit for collaborative handling of issues and reviewing code contributions in turn stem from open source projects. New tools are continuously introduced into the market, with the goal of supporting the developed product and the development process [24]. Since these tools have largely been about 'scratching one's own itch', the developers working on proprietary software have quickly realized that these tools really empower the developers. Hence, tools and processes made familiar by open source development are used to distribute the development effort globally, to manage quality, and to enable agile development, manifesting transparency in every development phase.

Furthermore, companies do not only apply open source tools and associated methods in their internal development, but also contribute directly to open source projects. To this end, big closed source players such as Microsoft, Google, Oracle, Facebook and Amazon are nowadays major open source contributors, with a large open source codebase of their own [25]. For many companies, this has meant a complete transformation from their organizational perspective and operational culture – so-called agile transformation [26], increasing transparency and empowering the developers in particular.

Finally, there are also more unscrupulous motives to embrace open source than those above. For instance, large companies may contribute to OSS just because they no longer see viable business around a product, to white-wash their public picture of becoming a Linux/OSS lover, or simply because they have to, as someone accidentally misused a GPL component.

4 How the Bazaar Turned into a Cathedral?

While anyone can start an open source project without commercial interests, the fact is that many of the projects work well because someone is funding the work. A common model is that an open source project is mainly a product of one company that is hosting the infrastructure needed by the project, contributing the biggest chunk and then inviting others to join in. Naturally, as the company

seats most of the maintainer spots, it has a wide influence on what gets in to the main branch of the project and what does not. Typically, the funder has its own interests, although based on contributions also other stakeholders' interests can be identified – for instance, with respect to the Qt framework and open source Qt Project, one can clearly see from the contribution timeline when eg. different phone vendors have had their time in adopting the GUI framework to their own platform-specific needs.

Today, it is easy to find big names in open source software. Linux has risen as the de-facto OS for clouds, servers, and mobile and embedded devices alike, and web browsers are predominantly open source due to the rise of Chromium next to Firefox, and Microsoft letting go of their closed-source implementation [27]. With Linux at its core, Android now has the largest market share of smartphone ecosystems. Recently Microsoft purchased the largest host of code repositories GitHub (https://blogs.microsoft.com/blog/2018/10/26/microsoft-completes-github-acquisition/). This rise to a key position in commercial settings has strengthened the corporate role in open source. As the direction of each project has significant commercial meaning, business stakes have become too high for the development to take unexpected turns, following individual developers' itch.

When growing in size and importance, many open source projects have started to lose their bazaar development model. Decision-making has become centralized, and processes are decided within a small group. For instance, in the Linux community, which has been an example of the centralized approach from its start, there is a well-defined governance model instead of everyone directly contributing to the code base. Similarly, open source foundations such as the Apache Software Foundation and the OW2 Consortium have set rigorous "cathedral-like" processes to developers who want to have their software in the foundation – a quality control process to follow, a program of incubation, and an "attic" to keep track of unmaintained projects (https://www.apache.org/foundation/how-it-works.html). In general, it is difficult to imagine any large-scale open source project that goes completely ungoverned.

Finally, stakeholder needs include ensuring the sustainability of the project as several stakeholders have significant business based on key open source software [28]. The recognition of IPR and the licence model are such key mechanisms. This has resulted in acts that are based on business reasons rather than the software itself. The selected licence itself acts as a cathedral-building mechanism and the licence and the licencing model can have major business impacts. Especially as not all licences can coexist within the same software, business drivers affect the software [29].

5 What to Look for Next?

During the recent years, we have seen open source and proprietary software come closer to each other and at the same time switch places. As a consequence

of this convergence, the hybrid nature of open source is thriving. When the open source community starts the development as a bazaar – like Linux for example has – they evolve towards cathedrals as they grow, gain business impact, and need to get organized to support the business stakeholders. On the other hand, communities that originate from business needs move to the opposite direction and adopt in part the bazaar model to get contributions from a wider contributor community. Similarly, the distribution of contributors in open source has turned towards corporate employed contributors instead of a buzzing group of volunteers, and in some of the open source projects, it is not possible for a volunteer to contribute any more [30] – you can fork individually but not join back as the company managing the software does not embrace contributions from other stakeholders, no matter how good they are. The issue is not only about controlling the individual developers' itch but also a wider perspective – what is the general direction of the project?

The modern way of accepting contributions is no longer through a monolithic project model but more from a distributed ecosystem marketplace resembling an appstore of modules/features. Perhaps the most vital communities that truly embrace participation of everyone are the likes of the NodeJS ecosystem (https://nodejs.org), where one can easily reuse or modify existing components, contribute new ones, and mash them together to create new systems [31]. These are modern bazaars where "capitalism of code" really rules – the most well-received modules/add-ons thrive and get additional contributors (if allowed), whereas individual hobbyist experiments can still co-exist but will eventually perish. Another example of this is the Microsoft VSCode (https://code.visualstudio.com/) where the actual open source project is the core platform, but the idea is actually not about the OS project (of developing a dull code editor) but about the very inclusive ecosystem where everyone can have a shot at creating an add-on. So, forget cathedral-like strict rules and CLAs (contribution license agreements) and bully-like maintainers, but just create your own "app" to the "store" and let's have the markets decide what will work.

Finally, changes in the value chain have had an impact to open source. Today, in many cases data, open APIs, and cloud services are far more important than the code, which has become a commodity. This attitude is backed by the fact that going for full open source means that one does not need to worry about issues such as licence compatibility, liberating companies from considering them.

6 Conclusions

By now it is obvious that open source software has realized its fundamental reuse promise. The many facets of software in general, together with the availability of readily reusable open source components in particular, have woven open source so deep in software industry that they are next to inseparable. The different kinds of software endeavors can start from opposite ends of being open or closed, but by now it seems that over time, the benefits of both worlds is something to seek for for both small companies as well as giant enterprises.

References

1. Raymond, E.: The cathedral and the bazaar. Knowledge, Technology & Policy **12**(3) (1999) 23–49
2. del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: The qualispo approach to oss product quality evaluation. In: International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. (2010) 23–28
3. del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: An investigation of the users' perception of oss quality. In: International Conference on Open Source Systems (OSS2010). (2010) 15–28
4. Rudzki, J., Kiviluoma, K., Poikonen, T., Hammouda, I.: Evaluating quality of open source components for reuse-intensive commercial solutions. In: 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, IEEE (2009) 11–19
5. del Bianco, V., Lavazza, L., Lenarduzzi, V., Morasca, S., Taibi, D., Tosi, D.: A study on oss marketing and communication strategies. In: International Conference on Open Source Systems (OSS2012). (2012) 338–343
6. del Bianco, V., Lavazza, L., Morasca, S., Taibi, D.: A survey on open source software trustworthiness. IEEE Software **28**(5) (2011) 67–75
7. Kilamo, T., Hammouda, I., Mikkonen, T., Aaltonen, T.: From proprietary to open source—growing an open source ecosystem. Journal of Systems and Software **85**(7) (2012) 1467–1478
8. Kilamo, T., Hammouda, I., Mikkonen, T., Aaltonen, T.: Open source ecosystems: a tale of two cases. In: Software Ecosystems. Edward Elgar Publishing (2013)
9. Sirkkala, P., Aaltonen, T., Hammouda, I.: Opening industrial software: planting an onion. In: IFIP International Conference on Open Source Systems, Springer (2009) 57–69
10. US Congress: National commission on new technological uses of copyrighted works. 1978. Final report of the National Commission on New Technological Uses of Copyrighted Works (CONTU)
11. Keplinger, M.S.: Computer software–its nature and its protection. Emory LJ **30** (1981) 483
12. DiBona, C., Ockman, S.: Open sources: Voices from the open source revolution. " O'Reilly Media, Inc." (1999)
13. Rolandsson, B., Bergquist, M., Ljungberg, J.: Open source in the firm: Opening up professional practices of software development. Research Policy **40**(4) (2011) 576–587
14. Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., Odenwald, T.: Open collaboration within corporations using software forges. IEEE software **26**(2) (2009) 52–58
15. Johri, A., Nov, O., Mitra, R.: " cool" or" monster"? company takeovers and their effect on open source community participation. In: Proceedings of the 2011 iConference. (2011) 327–331
16. Di Tullio, D., Staples, D.S.: The governance and control of open source software projects. Journal of Management Information Systems **30**(3) (2013) 49–80
17. Sadowski, B.M., Sadowski-Rasters, G., Duysters, G.: Transition of governance in a mature open software source community: Evidence from the debian case. Information Economics and Policy **20**(4) (2008) 323–332
18. Fitzgerald, B.: The transformation of open source software. MIS quarterly (2006) 587–598

19. Kanter, R.M.: When giants learn to dance. Simon and Schuster (1990)
20. Ackerman, A.F., Buchwald, L.S., Lewski, F.H.: Software inspections: an effective verification process. IEEE software **6**(3) (1989) 31–36
21. Paulk, M.C.: The capability maturity model: Guidelines for improving the software process. Addison-Wesley Professional (1995)
22. Mikkonen, T., Taivalsaari, A.: Software reuse in the era of opportunistic design. IEEE Software **36**(3) (2019) 105–111
23. Lokhman, A., Mikkonen, T., Hammouda, I., Kazman, R., Chen, H.M.: A core-periphery-legality architectural style for open source system development. In: 2013 46th Hawaii International Conference on System Sciences, IEEE (2013) 3148–3157
24. Sbai, N., Lenarduzzi, V., Taibi, D., Sassi, S.B., Ghezala, H.H.B.: Exploring information from oss repositories and platforms to support oss selection decisions. Information and Software Technology **104** (2018) 104–108
25. Hoffa, F.: Who contributed the most to open source in 2017 and 2018? Let's analyze GitHub's data and find out. FreeCodeCamp, Oct. 24, 2017, https://www.freecodecamp.org/news/the-top-contributors-to-github-2017-be98ab854e87/
26. Fry, C., Greene, S.: Large scale agile transformation in an on-demand world. In: Agile 2007 (AGILE 2007), IEEE (2007) 136–142
27. Warren, T.: Microsoft is building its own Chrome browser to replace Edge. The Verge, Dec. 4, 2018, https://www.theverge.com/2018/12/4/18125238/microsoft-chrome-browser-windows-10-edge-chromium
28. Nyman, L., Mikkonen, T., Lindman, J., Fougère, M.: Perspectives on code forking and sustainability in open source software. In: IFIP International Conference on Open Source Systems, Springer (2012) 274–279
29. Hammouda, I., Mikkonen, T., Oksanen, V., Jaaksi, A.: Open source legality patterns: architectural design decisions motivated by legal concerns. In: 14th International Academic MindTrek Conference: Envisioning Future Media Environments, ACM (2010) 207–214
30. Mäenpää, H., Kilamo, T., Mikkonen, T., Männistö, T.: Designing for participation: three models for developer involvement in hybrid oss projects. In: IFIP International Conference on Open Source Systems, Springer (2017) 23–33
31. Hartmann, B., Doorley, S., Klemmer, S.R.: Hacking, mashing, gluing: Understanding opportunistic design. IEEE Pervasive Computing **7**(3) (2008) 46–54