FLOSSing proprietary code

Seminarbeitrag, Beiträge zum Software Engineering

David Kaltschmidt (3782360)

December 5, 2011

1 Introduction

News headlines like "After seven years, Apple open sources its Apple Lossless Audio Codec"[1] always made me wonder. Being a company shrouded in secrecy about their products, Apple decided to release a piece of proprietary software under the Apache License to the Open Source community. While Apple made this move without any comment, they are not alone in opening up to the community.

Nokia started development on the Internet Tablet in 2005. Using a private-collective innovation[1] model the company enlisted the help of the Free/Libre/Open Source Software (FLOSS) community to build a commercial product. It based the development of this product mainly on open source software and made a large part of the research and product development transparent and accessible as a public goods innovation. Outside contributors involving firms and individuals, unpaid by Nokia, expended a significant amount of private resources on its development.

2 Paper: "Extending private-collective innovation: a case study"

M. Stuermer, S. Spaeth, and G. Von Krogh. "Extending private-collective innovation: a case study." In: R&D Management *39.2 (2009), pp. 170–191*

Since the Nokia Internet Tablet was being developed using both proprietary as well as open source software it is a case in point. Stuermer, Spaeth, and Von Krogh picked it as the subject of an empirical study on the incentives and costs of the private-collective model of innovation.

[1] Stuermer, Spaeth, and Von Krogh are using the term *innovation* in the tradition of Schumpeter [3, Ch 3.B, p. 83] as being a new combination of production factors (invention) and its immediate adoption.

2.1 Private-collective innovation

For any company or organization an innovation is advanced by an investment or action. In the world of technology the first two of the following models of innovation incentives dominate:

Private investment model In a private investment model of innovation, firms use internal processes to create ideas, knowledge, and technologies and commercialize these in the market place. Firms appropriate returns from private investment in innovation through intellectual property rights.

Collective action model Here the public subsidizes the innovators who, in turn, contribute to a public goods innovation. Public goods are characterized by non-rivalry and non-exclusivity in consumption. Innovations are made freely available to all as public goods. Here private companies have the option to free-ride on the public goods innovation. For example, a technology firm can commercialize and sell a product based on research done by a university without giving any of their research back to the scientific community.

Private-collection innovation In absence of a public subsidy a private firm can still decide to invest private resources to create public goods innovations. Such innovations are characterized by non-rivalry and non-exclusivity in consumption. Intellectual property rights are forfeited and the resulting innovation is offered to the public for free.

2.2 The subject of the study: Nokia Internet Tablet

The effects of the private-collective innovation model were studied in a single-case study following the development of the Nokia Internet Tablet. The study relied on semi-structured interviews, allowing participants the opportunity to narrate stories, provide anecdotes, and state opinions. As a second data source the project's developer mailing list was analyzed to determine the activity and size of the community.

Nokia's first steps Nokia started to develop devices built around the Linux kernel in 2000. These devices had no mobile phone functionality but were designed to allow mobile Internet access. For this they devised an architecture using open source components like GNOME, GTK and Gstreamer. Some of the components had to be extended in functionality or adapted to work on an embedded device. Nokia approached individual developers that were active in the open source projects that were crucial for the architecture and hired them as contractors to do the necessary implementations. They had to sign a non-disclosure agreement (NDA).

A prototype device was released in 2005 and 500 devices were sold at cost at a GNOME conference with proceeds going to the GNOME foundation. At the same time Nokia decided not to distribute the device through mobile shops. It was only available on Nokia's own website.

The Maemo community In 2005 Nokia consolidated their development efforts. It launched the Maemo platform which provided the whole software stack for application development for the handheld device. This included the open source projects together with binary files that form the operating system and the software of the Internet Tablet. The platform code itself was open source and was maintained by the Maemo community. The infrastructure supporting the community was split into a core and a garage part. Only Nokia employees and some Maemo developers are allowed to contribute to the core part. which consists mostly of the operating system. The garage part is open to the community and registered developers can distribute their software through it.

2.3 Benefits of the application of the private-collective innovation model in the Nokia case

The study revealed the following benefits which are in concordance to the findings of another study[4].

Cost of knowledge protection The effort necessary to keep the knowledge inside a company might outweigh the benefits.[2] Nokia decided to keep only 25% of its code closed source.[3] Another 25% of code being unmodified open source software. The rest, meaning half of the entire code base, were improvements and adaptations to open source software that was made publicly available. The decision not to release all source code was caused by a need for control over a certain aspect, e.g. the look and feel of the user interface, or a commercial licenses of a third party software.

The cost related to knowledge protection consisted mostly of the labor of checking the released open source code for revealed intellectual property or the use of a patent Nokia did not own. This cost can be substantial and it was not clear whether Nokia saved cost.

Learning from others' contributions This effect goes both ways as you can learn from the contributions of the community as well as learn in the process of creating something that you share. Nokia had started the Internet Tablet as a research project and had a lot of intimate knowledge of the whole architecture. Knowledge gaps were addressed by

[2]Eventually the knowledge will leak or be made obsolete, e.g. Apple's audio codec was reverse-engineered and an open source implementation was publicly available a year after Apple released its proprietary version.

[3]Including integrated third party software with commercial licenses.

either outsourcing small tasks to individuals or companies experienced in open source software development. With this dual approach Nokia learned from the contributions as it had to integrate them into the final product.

The biggest cost there was the time invested in reviewing the contributions and coordinating with the community. Another aspect of working with the open source community was to learn about its process where some projects have neither a roadmap nor hard deadlines. Nokia also made the self-confessed mistake of forking a whole project, GTK, and following a separate development stream. The resulting effort in moving changes between these branches made it increasingly difficult to benefit from other people's contributions.

Reputation gain A positive reputation can be gained from a commitment to open source. Volunteers contributing to the Maemo platform reported growing attachment to Nokia. There was also an economical effect as these volunteers bought the device knowing that they were buying into an ecosystem which they could improve and create applications for.

A second benefit from a reputation gain is that it attracts better developers. Nokia used this by recruiting developers based on their contributions or community standing. One of the interviewees was had an unfulfilling job and was working on the window manager used by the tablet in his spare time. Nokia recognized his contributions and hired him.

Widespread adoption The public availability of the innovation can lead to a fast and wide adoption, accelerated by network effects that may establish a dominant design. Since Nokia based its platform on widely used open source projects, contributors to these projects had no entry barrier to participate in the development of applications for the tablet.

When Nokia invited other companies to use its platform Intel joined and used the tablet's UI framework for their line of mobile devices. Even before that Nokia's efforts resulted in the creation of the GNOME Embedded Platform that can be reused and improved by others.

Lower cost of innovation Free access to innovations may incentivize a manufacturer to produce an item cheaper or to produce it with a higher quality, benefitting all consumers. Nokia helped create a solid and cheap operating system for mobile devices that can be used by other manufacturers. Here too, a leverage effect materialized. When, for example, Nokia pays a kernel developer to work on a problem, a solution may be found by collaborating with other developers who may be sponsored by other companies like IBM.

In the case of the Internet Tablet, an army of volunteers contributed in a variety of ways. In addition to application development, they filed bug reports, fixed the bugs and testes various peripheral equipment. Nokia regarded the application development as especially important because a successful application would have spurred sales of the tablet.[4]

Shared development cost Reuse by others invites further contributions which can improve the innovation and lower the cost of its creation. The quality of the own code can be raised and other existing technology from the community can be reused. For Nokia this translated into reduced fixed cost as they did not have to develop all of the technology by their own research and development team.

Furthermore, even essential applications like a mapping and navigation software were provided by the community. The use of the Maemo platform for the tablet also meant that no per-device license fees had to be paid by Nokia.[5]

Note: While the use of software that has been released under a copyleft license may lower the cost of innovation it may force the firm to contribute to public goods innovations (see also *Losing business secrets* and *Organizational inertia* below).

Faster time-to-market The use of modular open source software allowed Nokia to develop an operating system in a short time.[6] The faster development allowed for shorter release cycles. Each release of the software allowed Nokia to get feedback on its recent changes.

By collaborating with outside volunteers and contributors Nokia acted as a system integrator coordinating a loosely coupled network of component providers. However, sometimes the integration of different open source packages into one software proved challenging. Nokia solved this problem by contracting open source developers or small firms to integrate the software.

2.4 The costs

Obvious costs include the labor that went into all contributions to the innovation originating from the private company as well as the forfeit of intellectual property rights. Several studies uncovered hidden costs which, like the benefits, apply only to some types of innovation:

[4] In the case of Apple's iOS ecosystem of devices and apps it becomes evident how low cost of innovation can create a commercial success.

[5] If on the other hand Nokia had chosen SymbianOS as the operating system for its devices, a royalty of $7.50 per device sold would have to be paid.

[6] At the time of product launch, neither a product category nor a market for the tablet existed.

Lack of differentiation A competitor can easily devise a product with the same purpose, base the architecture on the same open source projects and reuse the communities adaptations and improvements. To that competitor Nokia is part of the community. This approach can be taken to the extreme when also the closed parts of Nokia's software are reverse-engineered and imitated, in effect, creating a clone. The cost hereby is the dwindling competitive advantage and lost sales as these replica products cannibalize each other in the market.

However, the above mentioned reuse is legal, and Nokia itself followed this approach. To limit the impact, Nokia revealed only the middle part of the software stack under an open source license. The lowest layer, the hardware-specific part, needed adaptation to Nokia's hardware, whereas the top user interface needed to retain a Nokia look and feel. These were kept under a proprietary license. In the interviews the community developers recognized this decision and signaled their understanding.

Losing business secrets Apart from business figures such as investments and numbers of sold devices it is difficult to determine what constitutes a secret that would have a value to a competitor. As Nokia's application of the private-collective innovation model was concerned with software development, only knowledge in the form of source code was shared. At the same time, development was spread among small firms who usually had to sign a non-disclosure agreement (NDA). Some motivated volunteers were even asked to sign an NDA or were simply hired by Nokia.

Lowering community entry barriers Nokia invested in the development of a Software Development Kit (SDK) that would help people inside and outside the company to get started quickly. To maximize participation, Nokia gave the SDK for free and provided the developers with upcoming platform releases.

Nokia also hired community organizers to act as the bridge between the internal developers and the external community members. They organized conferences where volunteers could learn about the technology. At the same time, Nokia offered away heavily subsidized tablets to very active community members. These were all real costs and could not have been mitigated. However these investments lowered future costs by increased participation, better knowledge diffusion and the mentoring of new community members both online and at the conferences.

Giving up control The cost of production increases with increasing dependency on external sources of technology. Choosing open source projects for the core of one's product means giving up control over the direction of the development. In the case of Nokia and GTK the firm invested heavily in efforts to make the the toolkit more mobile friendly, i.e. less resource hungry. According to the interviewees, Nokia's improvements and modifications paired with its respect for the meritocratic organization of the project gave it enough influence on the development. It should be noted that some community

members remained wary of Nokia's contracting of project contributors lest losing control on the community's side.

Two other events created extra tension. When Nokia decided to give write access to its repositories only to its employees, community developers felt excluded. This also meant that when valuable contributions to the open source projects were made *upstream*[7], contributors had to wait until a Nokia employee applied these patches. This was mild compared to the community outcry[8] when Nokia released version 4 of the Maemo platform only in binary format. Hinting at legal issues with the release, Nokia released the source code a few weeks later.

Organizational inertia The Internet Tablet included software written by third-party vendors. Therefor Nokia had to ensure not to infringe any intellectual property rights when releasing the Maemo platform. Every release has to go through a reportedly slow and bureaucratic review process. The complex internal processes of a multinational corporation can make it difficult to work with a highly dynamic community.

Secondly, some parts of the infrastructure like the bug tracker had to be duplicated since outsiders were not allowed access to Nokia's company network. At the same time, Nokia employees, blocked by the company's firewall could not access the Maemo developer IRC channel.

2.5 Summary

Nokia launched the Internet Tablet as a private-collective innovation project and as a lowcost probe. Rather than following existing market demand, Nokia targeted technology pioneers to find out who would use the Internet Tablet and how it would be used in real-life applications. Nokia opened up the product's software using externally developed open source technologies, allowed for and encouraged contributions by outsiders, and in the process created a new market for a product it had envisioned.

In my opinion, the biggest benefit seemed to have been the positive reputation gain by being labeled "open source friendly". This attracted developers who contributed to the platform, helped it grow and become increasingly appealing to other developers. Together with the lower development cost that comes with the reuse of software components, this is the only way Nokia could have accomplished this at such a low cost.

For a company as big as Nokia, the most important hidden cost was probably caused by the organizational inertia. Its internal time-consuming processes simply proved inadequate to collaborate with a very flexible community, resulting in frustration on both

[7] These contributions were also more numerous for usually a larger group of developers works on the original open source project.

[8] A common threat by the community is forking, i.e. continuing main development in a newly created branch with newly stated goals, in effect competing with the original project.

sides. This very fact makes me wonder how a much smaller company would have performed if they had set out to buy the hardware device from a manufacturer and develop the software for it.

Overall, the private-collective innovation model was attractive to Nokia as it kept development costs low, enabled external contributions and boosted organizational learning. On balance, the benefits seemed to have outweighed the costs as one interviewee summarizes:

> Some people might say that one of the problems is that you are leaking and giving out your secrets and so forth, but it's more like a trade-off. What is more important to you: to give some of your secrets an internal work-out or how much help in creating these products you get for free. I think, if you calculate, you are far more on the positive side when you decide to share.

On a side note, it is sad that the transcripts of the study's interviews are not available. Interesting questions to a Nokia manager included "When do you do things internally?" and "When do you decide to pay somebody for a certain development effort?".

3 Paper: "Selective revealing in open innovation processes: The case of embedded Linux"

J. Henkel. "Selective revealing in open innovation processes: The case of embedded Linux." In: Research policy *35.7 (2006), pp. 953–969*

Was the Nokia case presented in the previous chapter unique? Henkel [5] tried to find conditions across a host of companies under which the private-collective innovation model, he calls it *selective revealing*, is feasible. The study is limited to companies developing embedded Linux systems. Not all companies have the freedom to decide on what to reveal. A lot of OSS development is governed by the General Public License (GPL) requiring the company to reveal derivative works. But what if not all code is required to be revealed, what then are reasons for openness? And further, if companies had to reveal code, what proportion was revealed and what type of code?

3.1 Study of Embedded Linux

Embedded Linux is one of three most widely used operating systems for embedded devices ranging from VCRs to mobile phones. These device types are so different from one to another that no standard version of embedded Linux exists. Instead, developing for embedded Linux refers to the activity of extending or writing modules that make Linux more suitable for the respective device type. If derived works in the sense of the GPL are produced in the process, their source code has to be revealed by the time the device comes to market.

The study by Henkel [5] was done using a web based questionnaire, targeted only at embedded Linux developers and yielded 268 valid responses. Two out of five respondents indicated that they worked at a device manufacturer, one out of five said to be working for a software company specializing in software for embedded devices.

3.2 Ways to protect code based on OSS

Whether source code has to be released or not largely depends if it is based on software that was released under a license that forces it to be revealed, e.g. GPL. The source code of derived work based on GPL'ed software must be made available to all receivers of the software. In the case of embedded Linux this means that when a device comes to market, any buyer is entitled to obtain the source code.

Upon request only However, the source code does not have to be shipped with the product, instead, only be made available upon request. Furthermore, if the software is only used within the company or the customer decides not to request the sources, the code will stay secret. There is no section in the GPL requiring a company to make the code public. The firms are well aware if this fact so that, according to the study, almost half of the respondents choose to reveal code only to the customers.

Lead time Licenses like the GPL require the revelation of source code only on distribution. For products that have a long development cycle this means the code only has to be released once the development has been completed, giving it a considerable *lead time*. 45 % of respondents follow this approach.

Software architecture Depending on the licenses and the nature of OSS reuse as either a component, a module, a library, a driver, a separate program, etc. modifications have to be made available upon distribution. There is a long discussion going on about what counts as a modification of a software licensed under the GPL. For example, reusing a GPL'ed library puts the whole program under GPL because when it is run it includes the library code. Other OSS licenses do not have this restriction, e.g. the Lesser General Public License (LGPL) imposes its copyleft restrictions only on the software itself, say a library, and not on programs that link against it. After carefully considering all licenses of the reused software, a redesign of the code architecture allows to shift functionalities that require protection to either a very low or a high application layer. Nokia followed exactly this approach.

3.3 Share of revealed code

Asked what share of code that could potentially be useful to others is being revealed, hobbyists and developers at universities led the field with a sharing 92% of their code. For

commercial firms, the revealing behavior varies strongly, sharing 49% of their code but showing a standard deviation of 35%, a minimum of 1% and a maximum of 100%.

In addition, sharing seems to have become more common over time as 49% of all respondents said they reveal more than they did in 2000. Firms seem to appreciate the benefits that Nokia experienced while at the same time having become more skilled in devising an architecture that allows for selective revealing.

3.4 Types revealed code

The questionnaire asked if companies shared code of the following types:

Generic, other companies may use it 63% of hardware firms agreed, 85% of software firms

Important for the companies competitive position net agreement to share was only reported by software firms: 27%.

Differentiates the product same as above (30% of software firms), probably when it wants to create demand for related services or signal the firm's competence in the field

Specific to our hardware net agreement for hardware firms is 34.5%, probably also aimed at supporting the development of complementary products to the hardware

Specific to our application software still considerable at 28.6%

An open question to classify the revealed code yielded the following quotes stressing economical reasoning and lead time:

> "[We reveal code] where the cost of support is more then the profit made on sales. We then give it out without support."

> "Anything that will not give our competitors the possibility to copy our products one to one in a short time."

3.5 Reasons to reveal

Agreement on the twelve offered reasons to reveal code differed between hardware and software firms:

Hardware firms The main reason to reveal is clearly "because the GPL requires it". Of all the 12 offered reasons, rank 2 was claimed by "to appear as a good OSS player" which is considered a basis for reasons ranked 3–5: bug fixes by others, further development by others, and reduced maintenance effort. The high rank of these three confirms that firms do perceive technical benefits from the open source development process in embedded

Linux, and are willing to share their code with others in order to realize these benefits. After rank 5 the level of agreement drops sharply.

Software firms Marketing reasons ranked much higher: "revealing good code improves our company's technical reputation" ranked third while "visibility on the mailing list is good marketing" ranked 6. Henkel explains that software firms often act as suppliers to hardware manufacturers, performing commissioned development and therefor want to be perceived as experts in the field.

3.6 Revealing behavior

A multivariate analysis was done to show if revealing behavior is dependent on certain company characteristics:

Firm size The share of code that is revealed is larger the smaller the firm. This is consistent with the view that a small firm, due to resource scarcity, is expected to benefit more from external development support.

Firm policies The analysis could not confirm that firms that encourage their developers to reveal non-critical parts of the code do reveal more. This is quite odd. It suggests that the developers seem to act independent of the presence of such a policy. On the other hand, firms that have restricting policies and gave reasons to reveal—e.g. external development support, reputation and marketing—do not reveal significantly less suggesting that policies are mostly informed by these reasons.

Complementary assets Device manufacturers reveal significantly less than software firms. The manufacturers' devices do not seem to constitute a sufficiently protective complementary asset. However component manufacturers reveal more, probably related to the drivers that will enable other companies to develop assets.

Familiarity with embedded Linux The longer a company has been developing for embedded Linux the bigger the share of revealed code.

Reasons for sharing more: Development support and reputation Companies that consider either of these reasons important, clearly share more code.

Reasons for not sharing more: Marketing and the GPL Companies that reveal code for marketing reasons do not share more. An explanation could be that they do not share more in quantity but make what they share more visible.

Companies that give the GPL as an important reason to reveal code, do not share more. These companies seem to be reluctant to share code in the first place and only do so when compelled to by a license.

3.7 Summary

For a long time, efforts to profit from innovation focussed on exclusivity, i.e. the protection of innovation. Instead, Henkel suggests, companies should focus on the appropriation of profits from innovation itself. The rise of OSS has shown that freely revealing one's developments may be a sensible thing to do, in particular when community-based development is viable. This is favored when, as for embedded Linux, needs are strongly heterogeneous and the underlying technology is highly modular.

On the other hand, developing OSS does not imply free revealing. Commercial OSS development offers means to protect parts of one's own code, allowing for selective revealing. Across the sample of firms, on average about half the code developed for embedded Linux was being revealed while the other half remained protected. The share of revealed code was, other things being equal, higher for small firms, component manufacturers and firms experienced in OSS development.

Among the reasons to reveal, informal outside development support in the form of bug fixes, code improvement, and code maintenance figures prominently, and turns out to be a significant driver of revealing.

4 Conclusion

Nokia is indeed not alone in its application of the private-collective investment model of innovation. Both papers support the view that the model works well for commercial products that have a narrow focus and are very distinct from one another. Embedded devices show exactly this wide variety. As a result, enlisting the help of the community does not seem to lead to a competitive disadvantage.

Firms are deriving technical benefits from revealing code or "officially" enlisting the help of the FLOSS community as Nokia did. The benefits include higher software quality through reuse, lower development cost and the fostering of complementary asset development as in the Maemo community. For Nokia and the majority of companies in the sample in Henkel [5] the private-collective model of innovation seemed to work.

5 Presentation

The presentation will follow the outline of this document by using the development of the tablet as a story thread.

- introducing the Nokia Internet Tablet
- introduce innovation models
- describe study/interviews, why was the Internet Tablet development a case in point
- present benefits, costs, strategies to mitigate costs, peppered with anecdotes from the tablet development
- sum up how the application of the model worked out for Nokia
- what about other companies? introduce 2nd paper by Henkel
- present study with goal to find conditions under which application of the model is feasible
- generalize Nokia's approach of protecting some parts of the software
- present most important licenses (GPL, LGPL, BSD, MIT, etc.) and how they affect revealing vs. protecting, lead time and software architecture
- present findings: share of code, types of code, reasons to reveal
- present company characteristics and reasons that lead to adoption of the model/revealing
- close with summary of benefits and costs and strategies to mitigate these

References

[1] Ars Technica. *After seven years, Apple open sources its Apple Lossless Audio Codec*. URL: http://arstechnica.com/apple/news/2011/10/after-seven-years-apple-open-sources-its-apple-lossless-audio-codec.ars (visited on 11/08/2011).

[2] M. Stuermer, S. Spaeth, and G. Von Krogh. "Extending private-collective innovation: a case study." In: *R&D Management* 39.2 (2009), pp. 170–191.

[3] J.A. Schumpeter. *Business cycles*. Vol. 100. Cambridge Univ Press, 1939.

[4] E. Von Hippel and G. Von Krogh. "Open source software and the'private-collective'innovation model: Issues for organization science." In: *MIT Sloan Research Paper No. 4739-09* (2009).

[5] J. Henkel. "Selective revealing in open innovation processes: The case of embedded Linux." In: *Research policy* 35.7 (2006), pp. 953–969.

[6] J. Henkel. "Champions of revealing – The role of open source developers in commercial firms." In: *Industrial and Corporate Change* 18.3 (2009), p. 435.