A Case Study of Open Source Tools and Practices in a Commercial Setting

Vijay K. Gurbani, Anita Garvert
Lucent Technologies/Bell Labs Innovations
2000 Lucent Lane
Naperville, IL 60566 USA
+1 630 224 0216
{vkg,agarvert}@lucent.com

James D. Herbsleb
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268 8933
jdh@cs.cmu.edu

ABSTRACT
Commercially, many in the industry are using products based on Open Source. What have been missing are studies on if the commercial industry benefits from developing software following the open source development model. We present a case study that examines this issue by applying the concepts of the open source software development methodology to creating industrial-strength software. We conclude with lessons learned and open research questions.

Keywords
Open Source, Software Development, Session Initiation Protocol, Architecture

1. Introduction
Open source practices and tools have proven potential to overcome many of the well-known difficulties of geographically-distributed software development [9], and to allow widely distributed users of software to add features and functionality they want with a minimum of conflict and management overhead [11]. Some reports have appeared in the literature describing experiences with open source tools in an industry setting [6], and in fact there has been a workshop focused specifically on open source in an industry context [2].

It is not immediately obvious, however, that open source tools and practices are a good fit to a commercial setting. To be sure, open source software is used extensively in the industry, and the recent acceptance of Linux and the Apache project are excellent examples of this phenomenon.

Open Source Application Spaces: Fifth Workshop on Open Source Software Engineering (5-WOSSE) May 17, 2005, St Louis, MO, USA.

However, what needs further study is whether the industry as a whole can benefit from adopting the *methodology* of the open source software development. Is the open source development methodology conducive to the manner in which industry develops its software, or are there only certain industrial projects that are amenable to the open source development methodology?

In this paper, we report a case study on using open source development in telecommunication software. The project was an Internet telephony server originally built by one of the authors (VKG), and later administered as an open source project inside Lucent Technologies in order to speed development and quickly add functionality desired by different project groups who wanted to make use of it in their product lines. We describe the effort's experiences over a four-year period and present a number of lessons learned about how to make such projects succeed.

The rest of this paper is structured as follows: in section 2, we compile a set of characteristics that while common to all open source projects, may be exhibited differently under a commercial setting. Section 3 describes the software project we used in the case study. In section 4, we describe the initial development of the software and its use inside the company, the open source style setup and the experience as various groups begin to use and contribute to the software. We conclude with lessons learned and a discussion of further research questions suggested by our work.

2. Open Source Project Characteristics
While there is, of course no definitive set of characteristics that all open source project necessarily share beyond permitting legal and pragmatic access to source code, there are many practices which are common across a large sample of open source projects (e.g., [7]). Some examples

This work was supported in part by a grant from CMU Cylab and an IBM Faculty Award to the third author.

of how these practices seem potentially incompatible with commercial development are the following:

2.1 Requirements

Commercial projects typically devote considerable effort to gathering and analyzing requirements, in a process that often involves several disciplines including marketing, product management, and software engineering. Open source projects, on the other hand, rely for the most part on users who are also developers to build the features they need and to fix bugs. Other users generally have to rely on mailing lists and change requests [13] to communicate feature requests to developers, who then may or may not address them, depending on their interests and the perceived importance of the requests. In commercial environments, management, often operating through a change control board, makes decisions about changes based on business needs.

2.2 Work assignments

In firms, developers are generally assigned by management to projects and development tasks. There is usually an effort to match tasks with developers' skills, and often an attempt to match their interests if possible, but developers' choices are generally rather limited. In open source, developers typically choose what they want to work on. Generally, they begin building something they themselves need as users of the software. Those who continue to contribute tend to begin taking on jobs because of their perceived importance to the overall project [14].

2.3 Software architecture

It has often been argued that open source projects require a more modular architecture than commercial projects, and there is now some evidence that this is the case [10]. In fact, the architecture of the Netscape browser became much more modular after it was released as open source [10]. More generally, it is widely recognized that the structure of the organization is a critical determinant of the structure of the code [3, 8]. It is not clear how well architectures designed for a commercial environment will support the sort of collaboration that open source practices must support.

2.4 Tool compatibility

Most open source projects exist on their own, or coexist on hosting services other projects that have all decided to adopt the same set of tools. In commercial environments, however, the situation is generally more complicated. There is often a wider range of tools used, and it is not clear how to support open source practices in heterogeneous environments.

2.5 Software processes

Many commercial environments have various levels of defined processes, often accompanied by stage gate systems where projects are evaluated at various critical points along the development path. These process are generally seen as critical to assuring software quality. Open source, on the other hand, generally has very little in the way of formal process, and instead insures quality through the "walled server" [7], placing control over what goes into releases in the hands of a "benevolent dictator", or small group of proven technical experts. These two approaches may prove to be incompatible.

2.6 Incentive structure

Commercial development is profit-driven, while open source is driven by a complex set of motives, including the desire to learn new skills, the desire to create features one needs, philosophical beliefs about contributing to the general welfare, for enjoyment of the freedom to build what one wants, and sometimes as a political statement about commercial business practices. The practices that make the very different open source and commercial practices succeed may rest in complicated ways on the developers' differing motivations.

3. The Software: A Telecommunications Signaling Server

The specific software we use in our case study is a telecommunication-signaling server. The server is a faithful implementation of the Internet Engineering Task Force (IETF) Session Initiation Protocol (SIP [12]). SIP is an Internet telephony signaling protocol that establishes, maintains, and tears down sessions across the Internet. SIP is a text-based protocol that operates on the notion of a transaction. A transaction is a request issued by a client followed by the receipt of one or more responses (from that viewpoint, SIP is like any reply-response protocol like HTTP, SMTP, or FTP).

It is interesting to note that protocol development work in IETF has been compared to an open source project (pp. 47-52, [5]). IETF hold physical meetings three times a year; the rest of the time, parties interested in a specific work item in the IETF follow the discussions in a public mailing list. IETF is unique as a standardization organization in that no membership or dues are required for an individual to participate in the development of a protocol; all that is required is contribution in the form of time and running code. In the IETF, a protocol goes through many drafts; once rough consensus has been reached with a protocol and design tradeoffs are known, the protocol is released as a standards-track document. Interested implementers typically follow the work closely from its draft stage and produce a working implementation of the protocol as it progresses. By 2002, SIP became a Proposed Standard as described in [12].

By the early 2000, the telecommunications industry was starting to coalesce around a cellular telecommunications

architecture called the 3rd Generation Internet Multimedia Subsystem (3G IMS). IMS imposed additional requirements on SIP beyond what the IETF standards dictated.

A SIP system has many entities: proxy servers help end points (called user agents) rendezvous with each other; registrars exist to register user agents so they can be found easily. Integral to a SIP entity is the notion of a transaction. Thus, in a typical SIP software stack, a transaction manager (defined above) that is scalable and provides the many services that the standard requires is essential. Residing on top of the transaction manager would be specific SIP entities called transaction users: proxies, user agent servers, user agent clients, and registrars, are all transaction users.

The source code was written in the C programming language and Concurrent Version System (CVS) was used for source code control and versioning. The code executed on the Solaris and Linux operating systems. The original version of the software was written as a server, however, as we will discuss later, the code was re-factored to create a library, which currently hosts the server.

What we have described so far suffices as a technical context for the rest of the paper; however, interested readers can refer to Rosenberg *et al*. [12] for more information on the protocol and detailed workings of it.

4. The Open Source Experience

In this section, we will give an overview of how the code and the development process evolved, in order to clarify the experience base from which our lessons learned were derived.

4.1 Initial development

The initial work on developing the software was conducted by one of the co-authors of this paper (VKG) at Lucent Technologies, Inc. by closely following the work progressing in the IETF SIP working group (1999-2002). At this point in time, the development was mainly an effort lead by the author of the code and an additional developer. The author was in close touch with the work progressing in the IETF by contributing to and deriving a benefit from the discussions about the protocol. Once the code had enough features in it, it was taken to an interoperability event that is held periodically. The outcome of these interoperability events is to interoperate two or more independent implementations against each other to ensure that they work across a broad set of features specified in the protocol.

4.2 Ad hoc partnering

As the code grew stable and achieved feature parity against the functionality specified in the protocol, the author started to distribute the binary to a wider audience inside Lucent Technologies, Inc[1]. An internal website advertised new binary releases of the server for others within the company to download and experiment with. The maturity of the server implementation coincided with the burgeoning acceptance of SIP as a protocol of choice in the telecommunications domain (1999-2001).

As internal interest in the server grew, the capabilities of the server were demonstrated by closely partnering in an opportunistic way, with select groups. For instance, the author extended the programmability of the server by providing callbacks when certain SIP events occurred in the server (arrival of a SIP request or a response). Using this programmability, the server was tied to a collaboration- and presence-related framework that was the focus of research in other groups within Lucent Technologies, Inc. Partnering of this type benefited many research projects within the company. At this time, such partnering was mainly limited to integration with existing frameworks and jointly staging demonstrations.

4.3 User-initiated change requests

As the server matured, it moved beyond a research-only project and was being productized as part of a standard Lucent Technologies, Inc. offer. Initially, even though select groups within the company had access to the source code, there weren't any contributions from them beyond the users reporting their experience to the author. Most internal users were simply downloading the compiled version of the server and using it for their work. Expanding the class of users in this way created a positive feedback loop in which the original code author implemented new features these users needed. The author encouraged other users within the company to use the software and report feedback and wishes for new features. This communication was conducted in an ad-hoc fashion, primarily over email and an updated web page. Requests for new features were ordered according to the business needs of the group productizing the server and the research interests of the author (often time, luckily, these coincided).

As SIP continued to gain industry adherents and as the general field of Internet telephony became more important, the server was viewed as a critical resource by many groups. Certainly, having access to the source code of a standard compliant server was extremely advantageous, more so since the standards were in a state of flux as SIP further evolved to touch other aspects of Internet services such as instant messaging and presence. By 2003, the

[1]While the server was not made available for download outside the company, for the sake of interoperability, it was hosted on a machine accessible to the public. Implementers outside Lucent Technologies, Inc. can use the server to benchmark their implementation even today.

server's source code was studied extensively by other groups within Lucent Technologies, Inc. Requests started to arrive on evolving the server to serve as a framework for many SIP-related groups within the company.

4.4 Establishing open source development project

At about the same time that requests for product-specific changes began to accelerate, others within the company started to contribute code and ideas back to the author. The stage was set to enter the traditional open source development model, albeit within an industrial setting.

The author of the original code (VKG) assumed the role of a "benevolent dictator" controlling the code base to ensure that the contributions coming in and features that other groups were proposing to build into the code matched the architectural principles of the software.

The author re-factored major portions of the server code to create a transaction library that could be used by any project within the company (since all SIP entities need a transaction manager). Working in close co-operation with two other projects, APIs and interfaces between the transaction manager and the transaction users were defined for information to flow between the manager and the transaction user. Re-factoring the software in this manner was very successful and enabled rapid creation of user agents [1] that executed on top of the transaction manager. Since the user agents were using the services of a transaction manager that was already implemented and tested, the programmers of these user agents could concentrate on the task of implementing the specific behavior of the user agent itself instead of worrying about the details of handling SIP transactions and other protocol-related minutiae. The re-factoring has been so successful that the initial server now runs on top of the transaction manager as well. Other groups that want specific transaction users can build them over the transaction manager by simply adhering to the APIs and interfaces.

Following the open source model, the code was made widely available to any project within the company that had a need for it. The "many eyeballs" effect of open source development is well known (i.e., the code benefits from being scrutinized by a wider audience with different interests and capabilities). This effect exhibited itself in this specific project in many interesting ways:

- By studying the code, the performance experts suggested a list of changes that would optimize the implementation [4];
- API experts suggested a layer of API that would lead to a more programmable framework;
- Others who were working on a 3G IMS project suggested (and contributed) modifications that made the code compliant to that architecture;
- Others still ported the code to other operating systems such as Windows and pSOS (a real-time operating system).

There are three reasons why these groups contributed the changes. The first is that having a stable, standards-compliant implementation provided motivated individuals a test-bed to try out new ideas (for instance, a major contribution to the code was a technique to optimize the parsing step). Another very important reason was saving time by making the contribution part of the base software. Unless this was done the group may have to manage their contribution separately. This may involve merging their contribution to the base code each time a new release arrived. To avoid this, it was better to contribute the change. A third reason is that certain groups, having used the software, *wanted* to contribute something back.

One big advantage of using open source techniques is to allow other groups to examine the existing code and bring their unique expertise directly to bear. For instance, while the original author was well versed in the IETF standardization process, he found it too time consuming to keep up with the 3G standard as well. Thus contributions coming in from the 3G team reflected their expertise, and were a welcome addition to the code. The problem, of course, was managing this complexity.

4.5 Delegating responsibility

The synergy that resulted in the complexity lead to the replication of another well known phenomenon in the open source community: the role of a "trusted lieutenant." Management identified strategic personnel in different groups and assigned them to manage key portions of the code while working closely with the author of the original code. Currently, the code is being modified by two main groups, using two concurrent lines of development. To manage these lines, there are two independent source code repositories (using different source code management systems). The intent is to merge the code across both the lines of development such that there is only one code base (currently, a set of specific customizations are applied manually to the base software, a step that will be eliminated after the merge). Because source code management systems and tools vary among projects, and also because the build systems are often tied to the source code control system, it is still likely that different groups will use their own source code repositories; however, the source code available to all groups will be identical.

The open source development model has, undoubtedly, succeeded for this project. Currently, there are 18 different projects within the company using the server. Of these projects, 30% are actively contributing code (or have already contributed code). There are currently 17 active contributors to the code. Some of these are responsible for user agents running on top of the library, others work on

the transaction library itself and still others work on providing support in the form of applying customizations and distributing the code to different projects within the company.

5. Conclusion

Our experience from harnessing the open source development methodology has resulted in a number of lessons learned, which we present in this section. We are also, however, experiencing several ongoing problems that present opportunities for future research.

5.1 Lessons learned

First, management support for the "benevolent dictator" is essential if the project is to succeed. Keeping up with the changes being made to the code as new features are added and accepting contributions from the set of interested users is a time consuming task. The benevolent dictator should be the final arbiter on what goes into the code while preserving the architecture (although this is not always possible; see next lesson).

Second, unlike traditional open source, the benevolent dictator cannot be concerned solely with a personal vision, a vision of the developers and users of the software when making decisions about what features go in and how the software evolves. In a commercial setting, those features that attract the most paying customers must percolate to the top of the priority list. The benevolent dictator can still remain a powerful force for maintaining the conceptual and architectural integrity of the software, but business necessities must be respected as well.

Third, owning the source code and having many eyeballs contributing to it has made it easier to keep up with the numerous extensions to SIP. It is beyond the capability of one team to be knowledgeable in all aspects (for instance, the team that knows about performance optimization may not know too much about security). Having access to the source code is invaluable since different individuals contribute in different ways to the cohesive whole.

Fourth, if possible, independent strains of the software should be discouraged, or tracked carefully. One of the biggest challenges we faced was how to merge independent changes done across two development lines. Each line had features and bug fixes that the other one wanted.

Fifth, a well thought out code distribution strategy is important. In traditional open source, the recipient receives a tar file (or downloads the source tree) and proceeds from there. However, in a commercial setting, the distributed code has to fit in the load building strategy of a particular group.

Sixth, since the standards and the technology were rapidly evolving, owning the source code allowed the company to respond quickly to customer needs. The authors of the paper have witnessed many commercial companies who have purchased SIP stacks from third party vendors; in such cases, these companies have to depend on the release schedules of the stack vendors. In developing solutions in the Internet timeline, this delay can provide extremely costly. Identifying states of flux such as this should be a valuable guide to finding opportunities for internal open source projects.

And finally, it is important to move toward a common set of development tools, particularly version control and change management systems. Unlike traditional open source, the broader community of developers is constrained by the tool environments of their project work. Moving code among different version control systems in order to build a variety of products is a difficult problem, and introduces the temptation of maintaining separate forks for each project.

5.2 Open questions

The experience we have gained leads to yet more open questions. As more projects are using the software, each one wants to customize it in its own manner. It is a challenge to allow such customizations while still preserving the core architecture. It would be extremely valuable to improve our understanding of how to design architectures to support open source style development.

Another question concerns limitations of the open source development methodology. Can what we did at Lucent Technologies be replicated with any random project across all industries? The answer is that it depends. We succeeded due to the convergence of many external forces and ideas. The manner of protocol development in the IETF was a big impetus to our project since we essentially tracked the earlier drafts; i.e., our implementation matured with the standard. When we started our work, Internet telephony was not viewed as mainstream a technology that it has now become. We just happened to be positioned at the right cusp when the company was looking for a SIP implementation that was standards compliant and that it owned. It is not clear, in general, how and when to initiate a project that can serve as a shared resource.

We also had a significant pool of users who were interested and capable developers, which seems to be a precondition for a successful open source project. If SIP servers were simply a well-understood and stable commodity technology, product groups could simply use it out of the box. We speculate that open source will succeed where there is 1) a technology that is needed by several product groups (hence there is reason to pool resources), 2) the technology is relatively immature so that requirements and features are not fully known at the outset, but rather evolve over time, 3) product groups have different needs and specific expertise in customizing the software for their needs, and 4) the initial product has a sound, modular architecture so that it it feasible to merge all the diverse

changes into a single development branch. Hopefully, future research will shed light on whether these speculations are correct.

6. REFERENCES

[1] Arlein, R. and Gurbani, V., An Extensible Framework for Constructing Session Initiation Protocol (SIP) User Agents. *Bell Labs Technical Journal*, *9,* 3 (November 2004), p. 87-100.

[2] Broy, M., et al., Workshop on Open Source in an Industrial Context, http://osic.in.tum.de/

[3] Conway, M.E., How Do Committees Invent? *Datamation*, *14,* 4 (1968), p. 28-31.

[4] Cortes, M., Ensor, J.R., and Esteban, J.O., On SIP Performance. *Bell Labs Technical Journal*, *9,* 3 (November 2004), p. 155-172.

[5] DiBona, C., Ockman, S., and Stone, M., *Open Sources: Voices from the Open Source Revolution*. 1999, Sebastopol, CA: O'Reilly.

[6] Dinkelacker, J., et al. *Progressive open source*. in *International Conference on Software Engineering*. 2002. Orlando, Florida.

[7] Halloran, T.J. and Scherlis, W.L. *High Quality and Open Source Practices*. in *Meeting Challenges and Surviving Success: 2nd Workshop on Open Source Software Engineering*. 2002. Orlando, FL.

[8] Herbsleb, J.D. and Grinter, R.E., Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Software*, *Sept./Oct.,* (1999), p. 63-70.

[9] Herbsleb, J.D. and Mockus, A., An Empirical Study of Speed and Communication in Globally-Distributed Software Development. *IEEE Transactions on Software Engineering*, *29,* 3 (2003), p. 1-14.

[10] MacCormack, A., Rusnak, J., and Baldwin, C., *Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code*, in *Harvard Business School Working Paper*. 2004: Boston, MA 02163.

[11] Mockus, A., Fielding, R., and Herbsleb, J.D., Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, *11,* 3 (2002), p. 309-346.

[12] Rosenberg, J., et al., SIP: Session Initiation Protocol, IETF RFC 3261, July 2002, http://www.ietf.org/rfc/rfc3261.txt

[13] Scacchi, W., Understanding the requirements for developing open source software systems. *IEE Proceedings on Software*, *149,* 1 (February 2002), p. 24-39.

[14] Shah, S. *Understanding the Nature of Participation and Coordination in Open and Gated Source Software Development Communities*. in *Annual Meeting of the Academy of Management*. 2004.