DOI: 10.1145/1646353.1646392

BY VIJAY K. GURBANI, ANITA GARVERT, AND JAMES D. HERBSLEB

Managing a Corporate Open Source Software Asset

WE DEFINE *CORPORATE OPEN SOURCE* (COS) as applying the precepts and methodologies prevalent in the open source development community for creating industrial-strength software projects in a corporation for internal use. It may seem that open source style development - using informal processes, voluntary assignment to tasks, and having few financial incentives - may not be a good match for commercial environments. Our ongoing work, however, demonstrates that under the right circumstances, corporations can benefit from open source development techniques. We present two approaches to managing COS projects, and expand in detail on one of them. Our results indicate that open source approaches require significant adaptation to succeed in commercial settings. In particular, they require substantial support from business divisions within a corporation to successfully leverage the shared asset.

Our ongoing research has attempted to determine whether corporations can effectively leverage the open source development model to create and manage software projects inside the corporate domain.[3,4] We have observed how the precepts and methodologies of the open source development had to be adapted in order to create commercial grade software. In particular changes are required in order to accommodate a market-driven schedule and feature decisions that are not wholly amenable to an open source development approach. Our contributions in this article include describing two methods to effectively manage COS assets: an Infrastructure-based COS model, and a Project-specific COS model. We report experiences with the management aspects of the latter COS model, which includes our findings that this model requires a greater amount of support to get a new business division onboard when compared to the minimal support provided by traditional open source projects. However, the benefits of Project-specific COS outweigh the costs once the business division is fully on-board: the development costs are amortized over the number of divisions using the common asset, and the asset itself benefits from contributions from the expanded use.

Open source practices and tools have proven potential to overcome many of the well-known difficulties of geographically distributed software development,[5] and to allow widely distributed users of software to add features and functionality they want with a minimum of conflict and management overhead.[6]

Dinkelacker et al.[1] discuss Progressive Open Source as a set of tools and techniques for a corporation to host multiple open source projects within a company and between third parties. In the context of their work, our work on COS[3,4] corresponds to and furthers their work on what is referred to as "Inner Source" in their paper.

Our previous work[3,4] attempted to determine whether open source tools

and practices are a good fit for developing commercial-grade software especially in the light of the differences between the two camps: open source development is more iterative in nature when compared to the staged method of software development practiced at many corporations; the incentive structure between the two varies, as does the motivation factor; commercial software is usually characterized by process methodologies (CMMI, ISO, TL9000, among others), that are typically absent in open source development.. We reached the conclusion that certain commercial projects can indeed benefit from open source development methodology, especially those projects where:

- a technology is needed by several product groups (hence there is reason to pool resources),
- the technology is relatively immature so that requirements and features are not fully known at the outset (so there is a need to evolve continuously),
- product groups have different needs and specific expertise in customizing the software for their needs (so everyone benefits from the contributions of each group), and
- the initial product has a sound, modular architecture (so that it is feasible to merge all the diverse changes into a single development branch).

Furthering our previous work, the discussion in this article presents a management view of maintaining a COS asset. We discuss project management and planning aspects that are intrinsic to projects managed in this style.

Project Description

The specific software used in our case study is a telecommunication-signaling server that implements the Internet Engineering Task Force (IETF) Session Initiation Protocol (SIP7). SIP is a text-based Internet telephony signaling protocol to establish, maintain, and tear down multi-media sessions on the Internet. The development of the project evolved in four phases, mirroring its evolution from a research-only project to a corporatewide, common and reusable asset. A quick overview of the phases is provided next; interested readers are directed to for more details.[3]

Phase 1: Initial Development. The initial software was developed by one of the co-authors of this article (vkg) at Alcatel-Lucent by closely following the work progressing in the IETF SIP working group. At this 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 a number of interoperability events to ensure its compliance to the published specification[7] as well as other implementations.

Phase 2: Ad-hoc Partners. As the code grew stable and achieved feature parity against the functionality specified in the specification, the author started to distribute the binary to a wider audience inside the company. An internal Web site advertised new binary releases of the server to download and experiment with. 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 an event-based framework.

Phase 3: User-initiated Change Requests. Gradually, the server moved beyond a research-only project and was productized as part of the business division the author worked for. 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 leading to the implementation of 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 Internet telephony became more important, the server was viewed as a critical resource by many groups; the server's source code was studied extensively by other groups within the corporation. Requests started to arrive on evolving the server to serve as a framework for many SIP-related groups within the company.

Phase 4: Establishing a COS Project. 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 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.

It is important to point out that in corporate software development, each project has an affinity for a certain set of tools.[3] The set of contributors now adding features to the code were accustomed to their organization's development environment. Thus, some organizations took a copy of the source code archive and replicated it in their local software environment to closely model what the developers in that organization were accustomed to. Of course, since none of the organizations used the same source control software as the author, the source files were put under the source code control system of that particular organization. It was at this time that the concept of an independent and common source code repository was born. An open source group was formally created to co-ordinate the independent and common source code repository. This group, the Common SIP Stack (CSS) group, was headed by another co-author of this paper (agarvert).

The goal of the CSS group was two-fold: one, maintain an independent and common source code repository such that all projects within the company take their deliverables from the CSS group. This was not an easy task,

Table I: Comparison of traditional open source and Project-specific COS models.

	Traditional Open Source Model	Project-specific COS Model
Social and political infrastructure		
Decision making (vision, evolution, etc.)	• Benevolent dictator and trusted developers	• Chief architect and liaison
Load building	• Release manager	• Construction, verification and load bring-up engineers
Project management	• No explicit role	• Project manager
Technical infrastructure		
Packaging, releasing and cross-feature coordination	• Release owner	• Release-, delivery-, and feature- advocate
Feature design and review	• No explicit role	• Feature advocate
Code development	• Volunteer contributors, trusted developers	• Core team members and business division contributors
Work flow	• Ad-hoc	• Driven by business divisions
Funding	• Donations, dual-licensing	• Driven by business divisions in general, sponsoring division in particular
Formal support for end users	• Usually minimal	• Extensive
Licensing	• GPL, BSD-license scheme	• Dictated by corporate policy

and the reasons are enumerated.[3] The second goal was to evangelize the technology and the implementation by creating awareness of the resource within the company. The CSS group acts as a one-stop shop for all SIP needs that any project within the company may need. It was funded by multiple business divisions within the company and had a dedicated support structure.

Managing a COS Asset

There are two models for a COS program: in the Infrastructure-based COS model, the corporation provides the critical infrastructure (Web servers, download accounts, mailing lists, code archives, wiki-tools, etc.) that allows interested developers to host individual software projects on the infrastructure, much like the SourceForge system, which provides hosting capabilities for communities developing open source software. Individual developers who choose to make use of the infrastructure determine the level of support they are willing to provide to interested users of the software package. This model has been used successfully to provide discrete software packages (C/C++ compilers, shells, utilities) to the greater research and development community in a company (see Dykstra et al.[2] and http://www.bell-labs.com/project/wwexptools.) Dinkelacker's[1] "Corporate Source" program is another example of this model.

In the Project-specific COS model, an advanced technology group, or a research group funded by other business divisions in a corporation takes over a critical resource and makes it available for the larger audience. This model is appropriate when the software is more than a tool and instead is a primary technology of the company, is tied to revenue generating products for multiple business divisions, the technology is relatively immature and evolving, and when the cost of redevelopment outweighs the cost of commonality. This is the more challenging model and the focus of our work, which offers evidence that Project-specific COS provides a viable development model to manage overall development cost, provide the needed software support structure, and make effective use of geographically and organizationally diverse pool of expertise.

Roles and Responsibilities

To manage a COS asset, a support structure ("core team") must be put into place. Our work[3] mentions this core team, but did not touch on the specific roles and responsibilities of the team. Here, we identify the roles and responsibilities within the core team and then look at the work flow inherent in such a system. Table 1 shows a comparison of the Project-specific COS roles to equivalent roles in traditional open source projects as enumerated.[8]

The first role is that of a *liaison*. The liaison has overall responsibility for the open source product, manages all activities performed by the core team, and interfaces with each business division for new work requests. The liaison works with the *chief architect* (defined next) to review and prioritize the feature list, serves as the advocate for internally generated development requests, and communicates planning information to the rest of the corporation. The liaison is the marketing manager for the asset, encouraging new projects to integrate the asset, and encouraging software contributions, even from non-users. Since the Project-specific COS model is highly unconventional, a significant amount of time is spent familiarizing the business divisions with the partnership model and securing a commitment for their contribution to the asset and ensuring the support structure is in place. The liaison is also responsible for establishing the development and delivery environment for the software.

The liaison works closely with a *chief architect*, ideally someone who would have founded the asset and put considerable energy in creating an initial version of the software. This person typically should possess not only a good software engineering background but also an industry overview of how to position the technology to customers and how is the technology itself evolving (through standardization efforts like IETF or grass-root community efforts like the Java Community Process.) Management level support for the chief architect is essential as the architect is the final arbiter on what features should go into the software asset while preserving the overall architectural integrity. This role is analogous to the one played by a "benevolent dictator" in traditional open source development. An interesting dichotomy between traditional open source and COS manifests itself in this role: unlike traditional open source, the benevolent dictator cannot be concerned solely with a personal vision when making decisions about what features go in and how the software evolves. In a corporate setting, those features that attract the most external paying customers will 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.

The core team must also have construction, verification, and load bring-up engineers. These members interface directly with the business division using the asset to provide support for release management tasks such as compilation, load bring-up and verification. In addition, they also perform a variety of other support tasks such as maintaining the common asset's software and tool development environment, documentation, authoring release notes, and Web design. The minimization of this effort is essential (for example, we reused existing infrastructure as much as possible but kept the processes lightweight and automated.)

A full-time *project manager* is crucial to assist in release and load planning; to manage the tools used to define and track features; and to ensure process compliance, which are often endemic to the corporate world. Rounding up the core team are sets of *development engineers*. However, before we outline this role in detail, a detour is necessary to discuss the organizational dynamics of a COS asset, as they have a direct bearing on the role played by the development engineers.

A guaranteed pre-requisite to the Project-specific COS model is the identification of a sponsoring business division. The critical need for such a sponsor was detected repeatedly in our work and appears to be a recurring theme: During Phases 1 and 2 of our work, the business division that employed one of the co-authors (vkg) acted as a sponsor to what became a common asset within that business division. In Phase 4 of the project, the majority funding for the common asset (the CSS group) was contributed by a single business division. While it may be possible to initially start with minimal set of friendly users, a successful technology will invariably require a larger, more organized sponsor. The core team belongs organizationally to the sponsor, and an equitable funding model will need to be worked out between the sponsor and other business divisions within the corporation that are interested in re-using the shared asset.

In our work, in a pattern we suspect is typical, the asset started off as a research project with a lead group that saw a product opportunity. To minimize the overall support cost, the lead group contributed existing tools, support staff, processes, assistance and leadership to the project. By selecting a lead group with an existing infrastructure for project planning and software development, and a willingness to participate, only a very small investment was needed to establish the early shared asset. This was a tremendous advantage. The disadvantage, of course, was that the common asset initially became anchored within the lead group. Load planning, code development, feature content, even system bring-up was driven exclusively by the lead group. Backing away from the lead group as more users of the common asset were identified took careful communication and negotiation with the lead group. That is why it is so necessary to pay adequate attention at the onset to a long term vision of a common and shared software asset. This allows the weaning away process from the lead group to occur as transparently as possible; as this process succeeds, more business divisions start to use and contribute to the shared COS asset. Successful projects will continue to cater to the exclusive needs of the lead group while providing the asset to other divisions within the corporation that may not need the feature set championed by the lead group.

With the shared asset being used by multiple divisions in a corporation, we now revisit the role of the development engineers. The core development team is responsible for completion of full cycle development tasks, including architecture, design and unit testing, under the leadership of the liaison and working closely with the chief architect. One member of the core development team was tasked with a specific role of a release advocate to ensure that the code changes for all features were submitted on time, and kept track of all business division-specific impacts for the particular release. This role changed periodically to allow all the entire core team to benefit from the leadership and management experience.

Certain key members of the development team were assigned the specific role of a business division delivery advocate. Unlike a release advocate, delivery advocates were assigned to a business division that intended on using the common asset but were new to the concept of the Project-specific COS model. Delivery advocates assisted in the surprisingly difficult task of build integration to the business division. Build integration consists of working closely with the new business division to fit the common asset with the peculiarities of the build environment of the business division. Each business division has specific tools, processes, collected lore, and compilation dependencies that had to be taken in consideration to reuse the shared asset in that business division. Furthermore, the delivery advocate worked with the particular business division to ensure that its contributions to the common asset were assimilated in a manner conducive to the architecture of the common asset.

A member of the core team was also assigned as *feature advocate* to see a particular feature to completion. In this role, the feature advocate approves design documents, performs code inspections, and ensures that the change aligns with the overall software architecture. The release-, delivery- and feature-advocates collectively manage the contributions from other business divisions. These contributions, when built into the official product, became the responsibility of the core team, creating a compelling reason to get it right the first time.

Work Flow

Work requests, that is, requests from business divisions for aid in using the COS asset, requests from business divisions about supporting new features, bug fixes, etc., arrived from a multiplicity of sources. Each business division has idiosyncratic processes for feature creation and prioritization that must be accommodated in the Project-specific COS model. These work requests were managed as described next.

A Web-based work request process was put in place. All business divisions that required additional work to the shared asset inserted a request into a general list; those requests with some impact were then moved to a candidate list, managed by the project manager. Additional feature requests were also created by the core team based on the knowledge of upcoming changes driven by standardization efforts, software architectural limitations, or other requirements such as security and performance. These requests were

added to the candidate list, which was reviewed periodically (once a week) for commonality, estimation and commitment by the chief architect, the liaison, key members of the development team, and the project manager. At this time, it was important to identify common themes among the feature requests from different divisions, and negotiate with each business division to align on a common solution.

As expected, the candidate list would surpass the available resources. The assignment of staff to each feature was the responsibility of the liaison, who contacted the staffing managers within the business division requesting the feature. To avoid overwhelming the core team, some members of which were always needed for work that was not funded by a specific business division, we needed a way to share the work. We developed a workable solution: for substantial features half of the development effort was borne by the division asking for the feature and the remaining half was the responsibility of the sponsoring business division. This rule was not followed strictly, as some business divisions would contribute a portion of their technical head count to implement a feature, while others would require the sponsoring business division to allocate all resources. Regardless, finding an equitable funding model for the feature between the sponsor and the specific business division occupies a substantial amount of time of the liaison.

Conclusion

We have presented two techniques on COS. The lightweight Infrastructure-based COS model can be rapidly deployed to reuse common software tools across organization boundaries with little or minimal managerial overhead. By contrast, projects with certain characteristics we describe may benefit from the more involved Project-based COS model.

Our work on the management of Project-based COS has yielded two important insights: first, for such projects to succeed, it is imperative that they benefit from a large and organized sponsoring business division within the corporation that can act as a champion for the common asset. Second, and perhaps the more important finding is that formal support and ownership required as the common asset is integrated into products being created by other business divisions cannot be ignored. Unlike traditional open source development where interested parties simply download the source code, compile it, and over time gain expertise in it, Project-based COS leverages organizationally diverse staff to complete features. Thus, those business divisions with the highest integration of the common asset will contribute the largest effort, and in some cases fund a portion of the core team. Furthermore, because other business divisions in the corporation view the common asset as a core technology that they subsequently build into their products and then sell to customers, the expectations from the sponsoring division are much higher. Indeed, a certain amount of hand-holding is required to get new business divisions integrated into the Project-based COS model to the point that they become active users, and perhaps even active contributors, of the shared asset. C

References

1. Dinkelacker, J., Garg, P., Miller, R., and Nelson, D. Progressive open source. In *Proceedings of the 24th ACM International Conference on Software Engineering*, 2002, 177-184.
2. Dykstra, D., and Leto, K. NSBD and software distribution. *Dr. Dobb's Journal*, (Sept. 1998), 84-88.
3. Gurbani, V.K., Garvert, A., and Herbsleb, J.D. A case study of a corporate open source development model. *Proceedings of the 28th ACM International Conference on Software Engineering*, 2006,,472-481.
4. Gurbani, V.K., Garvert, A., and Herbsleb, J.D. A case study of open source tools and practices in a commercial setting. *Proceedings of 5th ACM Workshop on Open Source Software Engineering*, 2005, 1-6.
5. 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), 1-14.
6. 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), 309-346.
7. Rosenberg J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley. M., and Schooler, E. SIP: Session Initiation Protocol. *IETF RFC 3261*, 2002; http://www.ietf.org/rfc/rfc3261.
8. Fogel, K. Producing open source software: How to run a successful free software project. *O'Reilly Publishing*, October 2005.

Vijay K. Gurbani (vkg@bell-labs.com) is a distinguished member of technical staff in the Enabling Computing Technologies research domain at Bell Laboratories, Alcatel-Lucent, Naperville, IL.

Anita Garvert (anita.garvert@wowway.com) is a former technical manager at Alcatel-Lucent, Lisle, IL. She served as the liaison for the COS asset.

James D. Herbsleb (jdh@cs.cmu.edu) is a professor of Computer Science and director of the Software Industry Center at Carnegie Mellon University, Pittsburgh, PA.