Inner Source—Adopting Open Source Development Practices in Organizations

A Tutorial

Klaas-Jan Stol and Brian Fitzgerald, Lero—The Irish Software Engineering Research Centre

// Although inner source offers numerous benefits, many practitioners are unclear about what it is and how to adopt it. When adopting inner source, organizations should consider nine factors pertaining to product, process, and organization. //

OPEN SOURCE has had an enormous impact on the software industry.[1] Software development organizations have widely adopted open source software (OSS) in a variety of ways. Besides adopting OSS products, as either productivity tools or off-the-shelf components, numerous organizations have adopted open source practices to develop their software. This is called inner source because the software is sourced internally, although different terms have been used, such as "progressive open source" and "corporate open source."[2] Unlike with traditional approaches, developers of an inner-source project don't belong to a single team or department. Anybody in the organization can be a contributing member of this community, as either a user or contributor. Eric Raymond compared traditional software development approaches to building cathedrals, while calling open-source-style development a "bazaar."[3] So, you can view inner source as a bazaar within a corporate cathedral.[4]

Interest in adopting inner source is increasing because it can lead to benefits such as these:

- Increased software reuse through software products and components becoming available as inner-source projects for anyone in an organization to use.[5,6] (Normally, in many organizations, teams can't access other teams' source code.)
- Improved quality by leveraging Linus's law—"Given enough eyeballs, all bugs are shallow."[3,5,7]
- Open innovation by exploiting the broad expertise and creativity of the whole developer pool in an organization, rather than just one department or team.[7–9]
- Accelerated development with a community potentially as large as the organization's entire development staff, which can result in faster time to market.[4,10]
- Improved mobility of personnel as developers become more familiar with various software

0740-7459/15/$31.00 © 2015 IEEE

projects and the tools used in a central platform repository.[5]

Although these benefits are appealing, organizations might face challenges in understanding where to start or what to expect. So, practitioners and managers typically have many questions, such as these:

- What software product would be suitable for inner source?
- How does development of an inner-source project differ from that of a traditional software project?
- Why would developers want to start or get involved in an inner-source project?

This article aims to shed some light on these questions. On the basis of a comprehensive review of the existing research, as well as our hands-on experience in several organizations, we identified nine key factors for organizations to consider when adopting inner source.[2] The factors constitute a framework to understand inner-source initiatives.

To illustrate the factors, we present analyses of inner-source initiatives at Philips Healthcare, Lucent (which later merged with Alcatel), and Philips Research. These cases differ greatly in size, domain, and activity level and provide insightful examples of inner source. We collected the data for these case studies through interviews with key people involved and the initiatives' supporting documentation. Organizations interested in adopting inner source can use these factors as a lens to gauge where they stand and to guide implementation of their inner-source initiatives. An assessment of these factors can help identify a product suitable for inner source and potential barriers to inner-sourcing success.

The Philosophy and Practices of Inner Source

Several large organizations have adopted inner source over the last decade. An early study described Hewlett-Packard's experiences,[5] followed by experience reports from companies including Alcatel-Lucent,[11] Philips Healthcare,[4] IBM,[6] and SAP.[7] Each of these companies took its own approach to adopting inner source.

All development methods must be tailored to an organization's specific context. However, inner source isn't a defined methodology, such as, for instance, Scrum, which defines roles, ceremonies, and artifacts.[12] Instead, inner source is best captured as a philosophy, using those practices from open source communities that can greatly add value to an organization's development approach.

Common open source practices include[2]

- universal access to all development artifacts such as code and documentation, and allowing anyone to inspect and submit contributions;
- independent peer review of contributions by others in the developer community;
- informal communication channels, such as mailing lists and Internet Relay Chat (IRC) channels, to allow for ad hoc communication that can be archived for later reference;
- self-selection of tasks to let developers identify parts of the software they feel they can improve, or defects to fix; and
- early feedback and frequent releases to keep a project alive and quickly improving.

This list is by no means exhaustive, but these practices are a common subset in successful OSS projects such as the Linux kernel and Apache webserver.

> Inner source isn't a defined methodology, such as, for instance, Scrum.

The Nine Factors

The factors fall into three categories: product suitability, practices and tools, and people and management (see Table 1).

Product Suitability

The factors in this category are the seed product, its stakeholders, and the product's modularity.

The seed product. The first step toward an inner-source initiative is to select an appropriate seed product—an existing initial implementation of a software product or component. Similarly to projects in open source communities, starting an inner-source project from scratch is difficult. Without an initial vision of a project, it's hard to attract developers from across an organization to invest time and resources. Instead, it's much more useful to have a seed

product that can attract a developer community and grow to a successful inner-source project. This seed project must offer sufficient value to an organization. Starting an inner-source project around a new operating system or database management system is unlikely to attract many contributors because building such commodity software is wasteful.

Philips Healthcare's inner-source initiative started with a component suite built around the DICOM (Digital Imaging and Communication in Medicine) standard, which many medical-imaging systems use. Philips Healthcare develops various products in this domain, such as x-ray and MRI (magnetic resonance imaging) scanners. This component suite, which evolved to a platform for Philips' software product line, had significant business value, so developing it in-house was a logical decision.

Lucent's initiative involved an implementation of the Session Initiation Protocol (SIP) stack. The initial implementation was written by a single developer in the late nineties. At that time, SIP was becoming an important telecommunications standard, and there were no suitable implementations that were complete, affordable, and of high quality. So, developing the implementation internally proved a good decision because it represented significant business value to the company. Over time, the source code was made available to others in the company, attracting an internal community of users and contributors.[11]

Philips Research's initiative involved an implementation of the Philips File Standard for Pictorial Data (PFSPD), a file format for video sequences that had been developed internally. This technology wasn't commonplace when it emerged in the '90s, when PCs still had limited capabilities for video processing, which required special hardware. Philips Research used PFSPD for research on video-processing algorithms that could subsequently be implemented in hardware. Over time, hundreds of different algorithms were implemented, all using PFSPD.

The stakeholders. The seed product must have a variety of stakeholders so that it can benefit from contributions throughout the organization. This variety can result in contributions and requirements from different product groups or business units, introducing more diversity in the contributions. If only one team or business unit has a stake in a project, inner source will provide no real benefit because the project won't leverage the "wisdom of the crowd."

Philips Healthcare's software product line platform became an important part of a variety of products developed at the company. Because many stakeholders had an interest in this platform, pooling resources to further develop it made sense. Furthermore, given that the products had widely varying requirements, the platform also benefited from a range of improvements from the various stakeholders.

Lucent's SIP stack implementation also benefited greatly from the company's inner-source strategy. SIP is a text-based protocol and thus requires a parser to parse SIP messages. The SIP protocol is complex; although the initial implementation worked well, room for improvement existed. Developers from elsewhere in the organization offered such improvements. For instance, one developer with extensive parsing expertise contributed an improved parser implementation. Other contributors also improved parts of the code.

PFSPD's initial version was written by a few developers who identified a need to collaborate with US-based teams. However, those teams didn't have the necessary specialized and high-cost hardware infrastructure. Therefore, a new version of PFSPD for (at that time) newer platforms running Linux was necessary so that the teams could collaborate closely for research and implementation of video algorithms.

Modularity. Modularity is a generally desired feature of software and is particularly important for community-based development. It lets developers focus on learning a subset of the overall code to which they can then meaningfully contribute. It also facilitates parallel development. Different developers can work on different parts of the project at the same time, without any merge conflicts when they check in contributions at the same time.

Although an inner-source project should have sufficient functionality to be interesting enough to attract contributors, additions shouldn't compromise a module's cohesiveness. If this happens, the module might become too heavy and harder to reuse.

For instance, at Philips Healthcare, the initial component suite offered too many combinations. So, Philips architects evolved it into a more integrated and pretested platform that was highly configurable, thus making it easier to use correctly.

The PFSPD implementation benefited from a well-designed API early during the project. Additional utility tools (39 in total) were developed around the main library, such as converters, comparison tools, and viewers. Given that these were developed as separate tools, the tool

TABLE 1

The factors influencing adoption of inner source, using examples from three case studies.

Category	Factor	Observations from the three cases	Recommendations	Potential risks
Product suitability	Seed product	The products were the SIP (Session Initiation Protocol) stack, DICOM (Digital Imaging and Communications in Medicine) standard, and PFSPD (Philips File Standard for Pictorial Data) file format. Internally developed implementations offered common functionality.	Identify seed projects that implement a common functionality or a standard.	Starting an inner-source project with too many requirements up front might not attract sufficient developers to deliver the project soon enough.
	Stakeholders	A variety of stakeholders helped because different expertise benefited the projects. The stakeholders contributed additional features and improved algorithm implementations.	Identify a seed project that has different stakeholders to create a sufficiently large community of users and developers who all have an interest in the project.	Similar to conflicts that can arise among developers in an open-software project, differences of opinion might negatively affect a project. A large number of stakeholders might lead to conflicting requirements, possibly leading to personal conflict.
	Modularity	Modularity was important in all cases. The SIP stack was refactored, the medical platform was made configurable, and the PFSPD's functionality was divided into different tools.	Pay extra attention to the seed project's modularity and interface so as to facilitate reuse, expert contributions, and long-term compatibility.	A too-coarse granularity (too-large components) might become unusable. Packaging of components can inadvertently introduce new dependencies and risks for the main product. The product shouldn't be based on too many constraining assumptions about its runtime environment that would decrease its independence or modularity.
Practices & tools	Development practices	The projects adopted open source or hybrid approaches. All cases exhibited approaches that facilitated flexibility for contributors.	Facilitate a development process that encourages contributions and considers the project's and organization's constraints.	Owing to the project's complexity, developers might find it hard or uninteresting to contribute.
	Quality assurance	The participants performed peer review of contributions and responded promptly to user-reported problems.	Ensure quick turnaround of peer reviews, inspect problem reports, and support early adopting users, so as to resolve problems quickly.	If community members don't participate in peer review, quality might suffer and problems might not be identified until it's too late to fix them.
	Tools	The use of a common or compatible set of tools proved important.	Ensure that appropriate and common tool support is available, without people having to learn new tools.	Converting to a new toolset might affect productivity if developers aren't familiar with the tools.
People & management	Coordination & leadership	Coordination varied from traditional governance forms to more self-organizing forms in the smaller projects.	Recognize the project creator's ownership. As a project's importance and community grow, aim for coordination and management structures that maintain the contributors' interest.	Conflicts might arise about who should lead a project.
	Transparency	In all cases, the project's source code was available, as were one or more mailing lists (for developers and users). Sometimes the developers used a wiki to share knowledge.	Provide full access to source code to encourage code reviews and suggestions for improvement. Provide a wiki for knowledge sharing and a mailing list for internal communication that can be archived.	Developers and managers might be uncomfortable with sharing code, because of either the fear of losing control or the fear of personal assessment based on contributions.
	Management support & motivation	High-level management support and advocacy proved essential for success. Inner source empowered contributors because it let them fix local problems. In some cases, developers contributed in their spare time.	Advocate and lobby with management by showing how the project benefits the organization. Sustain the flexible nature of the project to ensure that contributors retain interest. Empower users by letting them solve their own problems.	If management makes strategic changes—for instance, withdrawing support to developers to work on projects they believe are important—developer motivation will likely drop, jeopardizing the initiative's sustainability.

chain exhibited considerable modularity. At some point, the developers found that maintaining and releasing these different tools was a burden and integrated them into a single command-line application while maintaining the original modular architecture.

> Peer review isn't unique to open source development, but the large scale on which this can happen is.

Practices and Tools

The second category of factors pertains to the development process and tools used in an inner-source project.

Development practices. Development of an inner-source project differs significantly from conventional approaches, including contemporary ones such as agile methods. Developers must be comfortable with a more flexible approach because new ideas (for example, features or improved algorithms) might be quickly turned into code, which the community of developers throughout the organization then reviews. This differs from conventional approaches that identify requirements before implementing them. In open source development, requirements are sometimes characterized as being "asserted after the fact"—features are added by developers who perceive certain missing functionality from their perspective.[13]

The degree to which organizations can adopt such flexible approaches will differ widely. Philips Healthcare's medical devices are subject to strict regulations (such as from the US Food and Drug Administration). It adopted a hybrid approach that let it comply with necessary regulations while gaining the flexibility offered by open source practices. In Philips Healthcare's codevelopment projects, business units (the "customers") and the core team (which manages the shared platform) worked together on newly identified features and requirements. Such codevelopment ensured sufficient domain expertise in the implementation and compliance with the platform's architecture. If a business unit urgently required a certain feature for a product release, it was free to make local changes to the platform.

An open source license would require any changes to be contributed back to the project. Inner source doesn't require this because licensing generally isn't an issue for internal development—the software is still proprietary, after all. However, it would be highly desirable to give back any changes to the core team. In that way, other business units can benefit from such additions, and the business unit doesn't have to carry the burden of maintaining its private features.

The PFSPD implementation was never an official project but rather an initiative by motivated developers to tend to their needs. Because development was extracurricular, work on PFSPD should be characterized as a series of development "bursts." When a new feature or improvement was necessary and time was available, developers worked intensively on the project. Before working on a feature, they usually announced this intention on the mailing list, which sometimes resulted in feedback on the proposed approach.

Quality assurance. Open source communities have developed QA practices that set them apart from traditional software development. For instance, peer review of contributions by the developer community effectively obtains feedback on contributions. Peer review isn't unique to open source development, but the large scale on which this can happen is, as reflected by Linus's law. Another motto common in open source communities is "release early, release often," which encourages solicitation of early feedback on new code.[3] Time-based, frequent releases also facilitate a stable rhythm of feedback. Such a consistent process eliminates the problem of developers rushing to have their new features included shortly before a new release because they're unsure when the next release will happen. With a reliable release schedule, developers know that, if a feature doesn't make it into the next release, they can fit it into the release after that, which will happen in a predictable manner.[14]

At Philips Healthcare, making new releases of the platform of several million LOC isn't trivial, and new versions typically are released twice a year. However, the core team regularly takes snapshots of the latest development version. Some teams use this opportunity for frequent integration to keep the feedback loop as short as possible. These teams found this proactive attitude worked much better than waiting for the regular, less frequent releases, at

which point much integration testing would be necessary.

In the PFSPD project, a key element of QA was that the developers immediately investigated any problem reports from users. This way, they quickly resolved issues, which also built the users' confidence in the project's quality.

Development tools. A seemingly trivial issue is the availability of a common set of development tools throughout an organization. However, many organizations have grown organically over an extensive time period, in which acquired businesses have become business units or departments, each with its own toolset. Different departments might also have a high degree of autonomy regarding what tools they use. These potentially heterogeneous and incompatible tools might impede the sharing of contributions or even the ability to build and run the software on certain platforms.[5,7]

Philips Healthcare is such a federated organization that has grown over time. It overcame the issue of disparate tools by using a standardized toolset from an external provider that offered a common development environment through a software-as-a-service approach. A local support team in the company helped business units deploy and use these tools—for instance, by providing training.

The PFSPD project benefited greatly from the common set of tools that were used early in the project. This proved particularly useful when one core team member moved to offices in the US. Having the infrastructure necessary for distributed development already available, development continued seamlessly. And, because of the new time difference, development could "follow the sun" during development bursts.

People and Management

The third category of factors pertains to the organization and its people, and addresses coordination and leadership, transparency, and management support and motivation.

Coordination and leadership. Inner-source projects need a more flexible approach to coordination and leadership compared to conventional approaches based on organizational hierarchies and roles. A key tenet of a bazaar-style approach is meritocracy.[7] Developers who contribute significantly to certain parts of the code, and thus have deep expertise, can become coordinators, or "trusted lieutenants," helping the "benevolent dictator" manage and coordinate the project. Coordination in bazaars is based on self-organization,[7] in which developers self-select tasks to work on, whether these are code contributions, defect fixes, or bug reports and documentation.

Many open source projects typically have no commercial aim (although companies are increasingly involved in developing some successful open source projects).[15] Nevertheless, it might be important to formalize a number of roles in the core team that's managing an inner-source project.[16] Which roles are necessary will depend on the organization's context, and they might emerge as needed. For instance, at Lucent, besides a benevolent dictator, a "community liaison" interacted with the various business units using the inner-source project.

The PFSPD core team comprised four key members, but other developers contributed too. The project's developer community was small, with a slightly bigger community of several tens of users.

Transparency. As we mentioned before, inner source is derived from the open source phenomenon, in which development is community based and the process is transparent and generally open to anyone who wishes to be involved. So, transparency is important but might not always be straightforward. Developers and managers might not be comfortable with sharing code and development responsibility.[5] However, providing full access to all development artifacts through supporting infrastructure such as a source code repository and a wiki for sharing knowledge is important. Projects should also maintain a mailing list for asynchronous communication and an IRC server for synchronous communication. Without such transparency, developers won't be able to "lurk" and will have no way to contribute to the project, which will greatly inhibit the inner-source initiative.

Without such transparency, developers won't be able to "lurk" and will have no way to contribute to the project.

At Philips Healthcare, all developers could access the source code of the SPL (software product line)

ABOUT THE AUTHORS

KLAAS-JAN STOL is a research fellow at Lero—The Irish Software Engineering Research Centre. His research interests include open source software, inner source, and agile and lean methods. Stol received a PhD in software engineering from the University of Limerick. Contact him at klaas-jan.stol@lero.ie.

BRIAN FITZGERALD is chief scientist at Lero—The Irish Software Engineering Research Centre and holds the Frederick Krehbiel Chair in Innovation in Business and Technology at the University of Limerick. His research interests include open source software, inner source, crowdsourcing, and lean and agile methods. Fitzgerald received a PhD in computer science from the University of London. Contact him at bf@lero.ie.

platform, stored in a central version control system. Furthermore, development became much more transparent, with a wiki server to facilitate organization-wide knowledge sharing and a mailing list through which developers could communicate and ask questions, much as how developers in open source communities interact. In fact, because the mailing list so successfully provided a Q&A forum, the initial help desk that Philips had set up internally was considered obsolete.

Transparency also played a role in the PFSPD project. Initially, access to the source code repository was limited to the core developers. However, PFSPD became an inner-source project once the source code was migrated to an internal SourceForge installation. The main communication channels were two mailing lists: a developer list to discuss daily development and progress and a list to support the user community. New releases were downloaded several tens of times.

Management support and motivation. Perhaps one of the most important factors in starting successful inner-source initiatives is top-level management support. Inner-source projects often start as grassroots initiatives by key individuals in the organization who strongly believe their organization can benefit from adopting open source practices. However, justifying work on inner-source projects can be difficult because it doesn't fit neatly in a strategic-planning roadmap. So, managers might consider any resources spent on such seemingly informal development programs to be wasteful and inefficient. However, once they realize inner source's benefits, management can become supportive.[7] This support must be complemented with a budget and resources.

Philips Healthcare's management made a strong commitment to its inner-source model. Although the model started out as a grassroots initiative, its benefits were soon demonstrated, and it earned top-level management's support and advocacy. Such support is important, but it's not enough to merely impose inner source.[4] Developers must be motivated to voluntarily participate in this model to make it successful. At the individual-developer level, having fun is an important intrinsic motivation widely found in open source communities. Vijay Gurbani and his colleagues described how this led some Lucent developers to work on the SIP stack in their spare time.[11]

At a higher organizational level, Philips has encouraged its business units to use, and actively engage in, inner source. Jacco Wesselius described how Philips Healthcare experimented with different business models to stimulate and encourage business units to contribute and engage in the development model.[4]

The PFSPD project was, as we mentioned before, never an officially funded project assigned to a specific team. The project emerged from the researchers' need to have PFSPD available on commodity hardware. However, because of this collaboration model's novelty, some contributors had difficulty justifying their time spent on the project because management didn't fully understand or appreciate their efforts or the nature of the inner-source initiative. For others, time spent on this project could be justified as a technology transfer activity. The general appreciation of the project's users further kindled the developers' motivation to sustain work on the project.

Whether developers get involved in an inner-source project also depends strongly on the corporate culture and norms. Some organizations have a highly traditional culture that emphasizes conformance with rules and guidelines, whereas others have

a more liberal culture that encourages taking the initiative.

Although we believe the recommendations we described are useful, merely following them won't necessarily result in a successful inner-source initiative. Other factors that can affect such a program's success might be at play, and more research is needed to better understand what makes or breaks an inner-source initiative. Furthermore, defining what makes such an initiative successful depends on the context. In the three cases we presented, inner source offered mechanisms that empowered developers. For instance, business units at Philips Healthcare were empowered by their ability to make local changes to the shared platform shortly before a release. Lucent developers were empowered to improve the code, and researchers at Philips Research could collaborate on their video algorithm research.

On the basis of our recent interactions with a number of companies in several countries, we believe inner source is gaining significant traction in industry. We join Wesselius in encouraging readers to share their experiences, challenges, and lessons learned in setting up inner-source initiatives.[4]

Acknowledgments

We're grateful to all informants of our studies for their time and enthusiasm. Special thanks to Bram Riemens for sharing his insights into the PFSPD (Philips File Standard for Pictorial Data) project. This research is supported, in part, by Enterprise Ireland grant IR/2013/0021 to ITEA2 SCALARE (Scaling Software), Science Foundation Ireland grant 10/CE/I1855 to Lero—The Irish Software Engineering Research Centre (www.lero.ie), and the Irish Research Council under the New Foundations program.

References

1. C. Ayala et al., "Five Facts on the Adoption of Open Source Software," *IEEE Software*, vol. 28, no. 2, 2011, pp. 95–99; doi:10.1109/MS.2011.32.
2. K.J. Stol et al., "Key Factors for Adopting Inner Source," *ACM Trans. Software Eng. and Methodology*, vol. 23, no. 2, 2014, article 18; doi:10.1145/2533685.
3. E.S. Raymond, *The Cathedral and the Bazaar*, O'Reilly, 2001.
4. J. Wesselius, "The Bazaar inside the Cathedral: Business Models for Internal Markets," *IEEE Software*, vol. 25, no. 3, 2008, pp. 60–66; doi:10.1109/MS.2008.79.
5. J. Dinkelacker et al., "Progressive Open Source," *Proc. 24th Int'l Conf. Software Eng.* (ICSE 02), 2002, pp. 177–184; doi:10.1145/581339.581363.
6. P. Vitharana, J. King, and H.S. Chapman, "Impact of Internal Open Source Development on Reuse: Participatory Reuse in Action," *J. Management Information Systems*, vol. 27, no. 2, 2010, pp. 277–304; doi:10.2753/MIS0742-1222270209.
7. D. Riehle et al., "Open Collaboration within Corporations Using Software Forges," *IEEE Software*, vol. 26, no. 2, 2009, pp. 52–58; doi:10.1109/MS.2009.44.
8. P. Copeland and A. Savoia, "Entrepreneurial Innovation at Google," *Computer*, vol. 44, no. 4, 2011, pp. 56–61; doi:10.1109/MC.2011.62.
9. L. Morgan, J. Feller, and P. Finnegan, "Exploring Inner Source as a Form of Intra-organisational Open Innovation," *Proc. 19th European Conf. Information Systems*, 2011, paper 151; http://aisel.aisnet.org/ecis2011/151.
10. F. van der Linden, "Applying Open Source Software Principles in Product Lines," *Upgrade*, vol. 10, no. 3, 2009, pp. 32–41.
11. V.K. Gurbani, A. Garvert, and J.D. Herbsleb, "A Case Study of a Corporate Open Source Development Model," *Proc. 28th Int'l Conf. Software Eng.* (ICSE 06), 2006, pp. 472–481; doi:10.1145/1134285.1134352.
12. B. Fitzgerald et al., "Scaling Agile Methods to Regulated Environments: An Industry Case Study," *Proc. 35th Int'l Conf. Software Eng.* (ICSE 13), 2013, pp. 863–872; doi:10.1109/ICSE.2013.6606635.
13. W. Scacchi, "Free and Open Source Development Practices in the Game Community," *IEEE Software*, vol. 21, no. 1, 2004, pp. 59–66; doi:10.1109/MS.2004.1259221.
14. M. Michlmayr, B. Fitzgerald, and K.J. Stol, "Why and How Should Open Source Projects Adopt Time-Based Releases?," *IEEE Software*, vol. 32, no. 2, 2015, pp. 55–63; doi:10.1109/MS.2015.34.
15. J.M. Gonzalez-Barahona et al., "Understanding How Companies Interact with Free Software Communities," *IEEE Software*, vol. 30, no. 5, 2013, pp. 38–45; doi:10.1109/MS.2013.95.
16. V.K. Gurbani, A. Garvert, and J.D. Herbsleb, "Managing a Corporate Open Source Software Asset," *Comm. ACM*, vol. 53, no. 2, 2010, pp. 155–159; doi:10.1145/1646353.1646392.