O'REILLY®

Getting Started with InnerSource

Keys to collaboration and productivity inside your company

Andy Oram

Getting Started with InnerSource

Andy Oram

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Getting Started with InnerSource
by Andy Oram

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editor: Nan Barber
Copyeditor: Holly Bauer
Interior Designer: David Futato
Cover Designer: Randy Comer

July 2015: First Edition

Revision History for the First Edition
2015-07-22: First Release

978-1-491-93758-7

[LSI]

Table of Contents

A Robust Approach to Team Collaboration

Inspired by the spread of open source software throughout the areas of operating systems, cloud computing, JavaScript frameworks, and elsewhere, a number of companies are mimicking the practices of the powerful open source movement to create an internal company collaboration under the rubric *InnerSource*. In these pages, you'll read about the experience of the leading Internet commerce facilitator PayPal and see how InnerSource can benefit engineers, management, and marketing/PR departments.

To understand the appeal of InnerSource project management, consider what has made open source software development so successful:

- Programmers share their work with a wide audience, instead of just with a manager or team. In most open source projects, anyone in the world is free to view the code, comment on it, learn new skills by examining it, and submit changes that they think will improve it or customize it to their needs.
- New code repositories (branches) based on the project can be made freely, so that sites with unanticipated uses for the code can adapt it. There are usually rules and technical support for remerging different branches into the original master branch.
- People at large geographical distances, at separate times, can work on the same code or contribute different files of code to the same project.
- Communication tends to be written and posted to public sites instead of shared informally by word of mouth, which provides

a history of the project as well as learning opportunities for new project members.
- Writing unit tests becomes a key programming task. A "unit test" is a small test that checks for a particular, isolated behavior such as rejecting incorrect input or taking the proper branch under certain conditions. In open source and InnerSource, testing is done constantly as changes are checked in, to protect against failures during production runs.

InnerSource differs from classic open source by remaining within the view and control of a single organization. The "openness" of the project extends across many teams within the organization. This allows the organization to embed differentiating trade secrets into the code without fear that they will be revealed to outsiders, while benefitting from the creativity and diverse perspectives contributed by people throughout the organization.

Often, the organization chooses to share parts of an InnerSource project with the public, effectively turning them into open source. When the technologies and management practices of open source are used internally, moving the project into a public arena becomes much easier.

Advantages of adopting an InnerSource strategy include:

- Code reuse across the organization grows immensely. Programmers from each team can understand the code and architecture of modules developed by other teams, and contribute code.
- This cross-team collaboration becomes relatively frictionless. Contributed code rarely has to be rewritten by the team receiving it, and no discussions are required at a high management level.
- Development becomes faster as programmers learn to use unit tests, code coverage, and continuous integration to remove bugs at an early stage of development. The written comments exchanged among team members, although taking up some time, more than pay for themselves by helping new programmers learn the system faster.
- Programmers learn to document their code better, both formally (as in-code comments and documentation) and informally (on discussion lists). The documentation provides a his-

tory of the project, and helps outsiders understand it so that more can contribute to it.

Where Open Source Principles Work

Before thinking about what InnerSource could accomplish for your organization, it's important to consider where the source of these practices—the free and open source software movement—has been found to work well, and where it has historically been less successful.

Cross-Organizational Collaboration

Open source software development crosses the boundaries of teams, companies, and nations, letting individuals around the world collaborate on such major computing advances as Linux and Apache. The stereotype of scruffy hackers creating code in a basement somewhere is outdated (if it was ever relevant): the major contributors to central open source projects such as Linux now work for major corporations.

A lot of important open source projects start out as the brainchildren of independent programmers, but they are frequently adopted by large companies. In other cases, open source projects emerge from within such companies, including PayPal. What remains crucial to their open source success is that people inside any single organization allow outsiders to play a major role in developing, testing, documenting, and promoting the software, creating as level a playing field for all contributors as possible.

The Difference Between Geographically Dispersed Development and Agile Programming

The cross-boundary activity of open source contrasts strongly with popular corporate development approaches, such as Agile and Scrum. The latter development methods were explicitly designed for teams working in a single office, communicating face-to-face. Not only must the developers in such projects work together, but they must meet with the software users and other stakeholders regularly. All this informal information sharing reduces the need for documentation and formal requirements. But such development meth-

ods don't work as well in environments larger than a single work site.

If Agile and Scrum pay a price in scalability, open source development pays a price in speed. Open source projects just can't move as quickly as dedicated teams working in a single office. This price obviously makes a big difference for start-up ventures with tight deadlines (sometimes insanely tight deadlines). The difference also has a subtle effect on the types of software that are well-suited to each development model, as you'll see in "Value for Software Infrastructure" on page 12.

Despite the limitations on speed of progress, many open source projects keep strict deadlines and have predictable releases. However, they often sacrifice features for predictability. Some enhancements that are planned for a particular release may have to be postponed in order to avoid slipping the deadline.

Agile programming does not have to be the enemy of open source development—in fact, the literature reports teams who combine the two.[1] But understanding their different requirements and assumptions can give you a better understanding of open source.

Continuous Testing and Development

In general, open source projects maintain quality and trust among collaborators by setting up a rigorous system of objectively testing each contribution. Many open source projects have adopted such commonly recommended practices as unit testing and continuous integration and have developed formal organizational models to ensure their use. Open source developers have turned quality control into a science.

Although test-driven development is not required for open source, unit testing is taken very seriously. Whether or not an open source project maintains a quality assurance team, each programmer is expected to write unit tests to assure the quality of her own code. Tools and protocols for committing changes require the tests to be run and produce a clean result before the code can be accepted into

1 Stol, KJ, P. Avgeriou et al. "Key Factors for Adopting Inner Source." *ACM Transactions on Software Engineering Methodology (TOSEM)* (2014): Vol 23, No 2.

the repository. The process offers assurance not only that the code works the way it was promised to work, but that it doesn't react badly with some other piece of code elsewhere in the project.

Modern techniques such as code coverage tools and static analysis are used less often in open source communities, and they've been adopted with dedication at PayPal.

Overall, testing and continuous integration play two important roles. Foremost, of course, they keep the product from breaking. But they also help to identify good coders who can be given greater responsibility for the project. For instance, getting a large number of commits accepted into the repository is usually a prerequisite to gaining the coveted status of a *trusted committer*. The trusted committers review and approve other programmers' work, and can make changes without needing such approval themselves. They also provide mentorship to promising contributors whose code doesn't yet meet quality standards.

The Importance of Documentation

Another tenet of open source development, stemming from the dispersion of its practitioners over geography and time, is full documentation. Open source projects tend to be weak on user documentation (a failing they share with most proprietary projects), but the developers obsessively write for each other about their assumptions, decisions, and implementations.

This practice represents another contrast with Agile programming, which calls for some documentation but generally favors "working software over documentation" (an oft-quoted clause from the Agile Manifesto). It should be noted that working source code is also the platinum standard in the open source community (as well as standards bodies dominated by developers), but these communities still consider it important to document what has been done.

Every facet of communication in open source software is written. Comments on GitHub are a key driver of many projects. Open source projects depend on mailing lists for discussions leading to all decisions, and you often hear, "If it's not on the mailing list, it didn't happen."

One value of documenting decisions and implementation details is that a history of the project is created for anyone new who wants to

join. Anyone who can take the time to peruse the discussion archives can pick up the project's culture and best practices.

Every collaboration assumes that participants share a common language. Given the geographic diversity of open source projects, of course, English has become the *lingua franca* practiced by all programmers (although there are significant open source projects in other languages as well). Making decisions through written communication plays a democratizing role here, because more people can learn to read and write a foreign language than learn to speak it fluently. These people could participate more in open source projects than in tightly knit teams using a language they don't know well.

Value for Software Infrastructure

Open source software has traditionally worked well for lower layers of the software stack: operating systems and hypervisors, tools used by programmers such as compilers and editors, security software (which benefits from open code reviews to detect weaknesses), and other things tucked away where the end user doesn't see it.

In contrast, the user interface (UI) has proven stubbornly resistant to open source development. It's hard to point to an open source end-user tool that has achieved mass adoption. Mozilla Firefox is a rare example.

Looking back at the contrast between open source and Agile development ("The Difference Between Geographically Dispersed Development and Agile Programming" on page 9) gives you a clue to the reason for this lack of success. The Agile model has been widely adopted because it keeps programmers in close contact with users. Developers get user feedback and can start working with it in a matter of days. Most open source projects involve end users in relatively old-fashioned ways, such as through alpha and beta releases.

The emphasis on unit testing ("Continuous Testing and Development" on page 10) also marks open source as appropriate for infrastructure. About five years ago, Agile expert Mike Cohn described a *test pyramid* that puts various layers of infrastructure underneath a small user interface layer. He mandated unit tests for as many layers of the software as possible, reducing dependence on functional testing.

The UI layer, by contrast, is very hard to check through unit testing, and these tests show decreasing reliability and value at that layer. Here's where functional testing enters to check the overall operation of a system and ensure that each action taken by the end user produces the desired result.

Thus, the testing process that undergirds quality and trust in open source is weak at the user interface level, which may be another reason that open source tools are less popular at that level.

Finding the Right Level for Open Source

Open source and free software has traditionally been contrasted with proprietary software (especially Microsoft Windows and the Oracle database). A more common approach to proprietary development nowadays is software as a service (SaaS). With this software delivery model, you can use the guidelines in this section to determine what should be developed in an InnerSource or open source manner, and what to do in a more closed manner using Agile or some other team-oriented methodology.

Lots of companies mix these models. Such companies could be called "closed core," because they keep the software critical to their business behind SaaS, but freely share infrastructure software. This arrangement benefits the wider programming community, makes it easier to recruit and train employees, and brings them the benefits of outside contributions.

How PayPal Adopted InnerSource

PayPal's path to InnerSource involved a series of historic, large-scale corporate decisions. It adopted the model as one of several shifts consciously made in tools and corporate culture, as is often the case.

In PayPal's case, the shifts that preceded the adoption of InnerSource included:

- A search for technologies that would promote faster development (replacing Java with JavaScript and Node.js on many projects, for example)
- A consequent interest in a better understanding of the open source communities and development models tied up in the newly adopted technologies

- Use of GitHub for collaboration both internally and externally
- A heightened concern for quality

PayPal, therefore, adopted some open source technologies and even open sourced some of its own code before experimenting with the InnerSource model. Other companies may take the reverse route: they may try out the tools and practices of open source within their own walls before producing any open source code, although that route is generally thought to be more challenging. In either case, a familiarity with open source tools, along with sites such as GitHub that facilitate collaboration in an open source manner, is crucial.

Although the various changes at PayPal took place together and clearly had impacts on one another, the next few sections will describe each change individually, focusing on each one's particular activities.

Starting at the Edge

InnerSource at PayPal started with Regional Sales Engineers, working outside the United States, who modified user-facing code to support local usage preferences and sometimes to support regional promotions. These programmers could not get their changes accepted by the core teams in the timely manner required. Often, these changes required the intervention of a VP somewhere to demand that they be merged in, after which an embattled core engineer would rewrite the submitted changes before merging them.

At the same time, inadequate documentation was being filled in through email exchanges carried out on an urgent basis between developers on different teams. Although these messages contained valuable information, they were unseen by most developers. Many regional organizations independently started efforts to cut and paste these messages into wikis.

A number of regional engineers were flown to the San Jose headquarters to spend a month learning directly from the core teams, but that method of bootstrapping trusted committers did not spread knowledge widely enough and was not worth the investment. No written documentation developed from it. After the engineers went back to their home countries, they were forced by distance to seek core mentorship through written queries.

Eventually, it was suggested that InnerSource might be a better way to go. Teams found that InnerSource really streamlined their projects, and the trusted committers on the core side found the process much better. Remote engineers could pull code samples from GitHub.

In addition to learning how to work with regional collaborators better, the core teams started to get clues (through their mentorship of regional contributors) about where to refactor core code to increase modularity and understandability. They were also able to notice patterns between work done in different regions that allowed them to promote experienced regional engineers as mentors to their peers in other regions.

A Speedier Development Process

PayPal started life with a relatively monolithic C++ platform. Although some legacy elements are still in C++, new development for a time was done in a more modular fashion using Java Spring. Even after this shift, as developer Jeff Harrell put it, a lot of "tribal knowledge" was embedded in each product. It took several weeks to roll out even a small change, and each new hire required a six-week training period.

Three years ago, PayPal made a major shift by adopting Node.js. According to Poornima Venkatakrishnan, a Node.js developer at PayPal, the platform was used there first for prototyping. Happy with the results, developers wanted to deploy it in production. First they tried an experiment where teams developed the same functionality side-by-side on the Java Spring platform and on Node.js. The company compared the platforms on the basis of development time, number of lines of code, and number of engineers required for development. Node.js was an obvious winner.

The announcement that PayPal was adopting Node.js worried many programmers, especially those who remembered the change from C++ to Java as long and grueling. However, PayPal demonstrated that the change to Node.js was completely different. It scheduled just two days for the transition training. According to Harrell, participants quickly realized they were entering a new, vibrant, and exciting world. PayPal was also lucky to have on staff a leader in the field of JavaScript, Douglas Crockford, to do the training. (Crockford is

the author of a highly popular book, *JavaScript: The Good Parts*, and creator of an O'Reilly JavaScript Master Class video.)

Much new PayPal feature development still uses Java, but those teams have adopted the InnerSource practices that were pioneered by the Node.js teams.

Engaging with Open Source

Training for the move to Node.js was quick and painless, because so much instructional material was available online and because adoption of the software required less customization than the previous move to Java. Essentially, two days of training was enough to prepare staff to continue their own learning processes.

According to Harrell, it took some time to convey to PayPal programmers that they could simply search for basic information on sites such as StackOverflow (or just use a search engine) instead of asking senior programmers on an internal PayPal mailing list. But eventually the queries for information broadened into a wholehearted engagement with open source communities.

Most companies find that adopting open source technologies—which may originally be attractive simply because they're high quality and free of charge—leads to a sustained entry into the communities that produce the technologies.

This happened to PayPal when they adopted Node.js. By now it is a pretty mature technology. Some commentators even suggest that Node.js has passed its peak and that JavaScript programmers are looking at alternatives. (The pace of change in open source technologies, as you will see later in this section, alters decision-making about which ones to adopt.) But PayPal came in at an early stage, where there was plenty of room to develop useful tools for the Node.js community. PayPal also became a very active member of the Node.js foundation and the ECMAScript Technical Committee 39.

To promote the twin goals of programmer self-education and immediate productivity among new hires, PayPal enthusiastically brought other popular open source tools along during the move to JavaScript and Node.js. For instance, Selenium is widely used for testing, Jenkins for continuous integration, and such tools as TestNG, JUnit, and Mockito for Java testing. Along the way, PayPal developed and open sourced a number of tools compatible with the open source

components, such as Nemo (originally written by Matt Edelman), SeLion, and Illuminator.

Since PayPal adopted Node.js when it was early in development and lacked certain tools, the company developed Kraken, a framework for controlling Node.js architectures. Harrell says the team decided to develop the framework as open source to make sure it was of general value, free from the "tribal" knowledge that Harrell had seen in other PayPal code. Kraken became what is probably their most successful open source project.

Venkatakrishnan lists the following open source practices that entered PayPal at this time:

- Setting high quality standards. Before code is committed, code coverage tests must run on at least 90% of it. When someone sends a commit, it triggers an automatic rebuild to make sure it's of sufficient quality to be merged.
- Requiring documentation for all code in the repositories themselves.
- Conducting all discussions on GitHub to enable collaboration and to facilitate input from people outside the team. There's a private GitHub repository for in-house projects as well as public GitHub repositories for open source code.
- Making everyone feel they can innovate.
- Teaching engineers that not all expertise needs to come from one team. With open source software, help can come from anywhere in the world. Also, contributing to public open source projects adds more visibility to the company and shows its commitment to the community.
- Cultivating pride in the work done in teams, encouraging them to speak about it at conferences and write about it in PayPal's engineering blog.

Harrell points out that the higher one goes in a software stack, the more volatile the technologies are. Change in software development is ongoing, especially with open source technologies. So PayPal encourages a team to constantly evaluate their technologies. Continuous learning is expected of each developer.

Inevitably, the evaluation of alternative tools leads to a diversity of tools from one team to another. Developer Matt Edelman points out that this phenomenon is particularly common in open source,

because developers like to develop tools along the lines suggested by Unix patriarch Ken Thompson: many small programs working together.

Thus, teams use a variety of cutting-edge technologies such as React and Angular. Programmers are expected to do a toy project to try out a tool before recommending it, and to compare alternatives carefully before making a choice.

Open source architectures often feature plug-ins to permit alternate implementations of technologies that extend their functionality. Edelman says that PayPal develops some Kraken plug-ins internally (through the InnerSource model) and others as open source.

GitHub Collaboration

One company holds a uniquely important place in modern code development: GitHub. Originally a SaaS platform to make life easier for developers using the Git version control software, GitHub evolved into a sophisticated collaboration platform with tools that developers find indispensable. Two resources on this topic include the book *Git for Teams* and the Collaborating with Git video.

In addition to putting code for open source projects such as Kraken on GitHub, PayPal has set up an internal GitHub Enterprise repository so that programmers within the company can collaborate exactly as public GitHub users do. Code reviews, commits, and tests take place in an open source manner, but on the internal repository. Most crucially, each team gets accustomed to accepting new code from PayPal programmers outside the team.

Before the move to GitHub and InnerSource, according to Edelman, PayPal programmers felt competent to contribute only minor bug fixes to another team's work. If they contributed larger chunks, the team accepting the code would scrutinize it and often do so much rewriting that there was little advantage over doing the code from scratch. High-level managers would often have to get involved to negotiate the acceptance of code between teams.

These days, substantial new functionality can be checked in with no rewriting. One reason is the new focus on documentation. Edelman stresses that InnerSource projects must rest on a broader programming base than a few privileged programmers who deeply under-

stand the system. Anyone can study a project and suggest major changes.

Also thanks to documentation (both in the code and the GitHub comments), developers come to recognize the need for architecture changes. If you have to explain four times to various people why something is complicated and counterintuitive, you start to think about changing it.

Empowering programmers across the country also promotes greater intellectual growth and job satisfaction. Programmers think more comprehensively about the design of the code, while learning new skills in doing code reviews, testing, and writing documentation. Edelman says InnerSource "raises everybody's game."

Edelman mentions one security-related bug that turned up in the core infrastructure module. One function wasn't following the specification (RFC) precisely. At first a lot of email was exchanged, with people outside the infrastructure team pressuring them to do a fix. But then one outsider—for whom security was not a specialty—took a look at the RFC and realized he could handle the fix himself. He turned in a fix that worked, and it was quickly merged by the core infrastructure team. This anecdote illustrates two principles: the importance of programmers taking initiative, and the value of following well-documented standards.

Quality Improvement

At the same time that PayPal was moving large parts of its platform to Node.js and adopting open source practices, it made a commitment to better quality. According to Quality Assurance leader Doug Simmons, steps toward this goal at PayPal included:

- More unit testing
- Continuous integration
- Code reviews
- More code coverage reports
- Static analysis

Not all conversations have to take place in public, but the comment process on GitHub has improved quality in several ways. Trusted committers effectively act as mentors for newer programmers hoping to get their code accepted into the repository. The back-and-forth comments between submitters and trusted committers are

educational for both the submitters and other programmers who can watch these dialogs and learn from them.

Culture Change

A corporate move as significant as adopting InnerSource calls for sensitivity in handling employee fears and expectations. PayPal hired outside experts in open source development to aid the transition.

Observers from many companies seem to agree that the most important cultural change is to give employees the confidence to contribute code to other teams. Full documentation and good mentoring can achieve this—but as Edelman points out, staff have to change their habits from complaining about bugs to going in and making fixes.

Harrell remembers persuading team members to let other PayPal programmers outside the team make contributions. The team members commonly objected that they'd end up spending all their time vetting outside code and not writing their own. In a study of another company, programmers worried about the burden that new contributions would add to future maintenance.[2, 3]

As you know by now, these drawbacks proved less fearsome than expected, because the rigorous testing and build process ensured the quality of contributions. In any case, Harrell told them, vetting contributions required sophisticated skills of its own. And the contributions multiplied the value of the code created by the team.

Edelman writes, "Module authors are expected to curate their software by being open to feedback and changes from the community, while enforcing quality and consistency standards." He says developers have evolved from complaining about the task for writing tests or doing code reviews to insisting on them.

In the quality arena, when programmers submit InnerSource bug reports, they go initially to the person who is responsible for the

---

2 Ibid.

3 Stol, KJ, AB Muhammad, et al. "A comparative study of challenges in integrating Open Source Software and Inner Source Software." *Information and Software Technology* 53 (2011): 1319–1336.

module. This programmer may then pass them on to the original contributors of the affected code.

Modular architectures and well-defined APIs have also been identified by many programming teams as crucial to encouraging contributions from outside the team.

Because InnerSource development practices are essentially the most popular open source practices, open sourcing a project that was developed inside the company is fairly easy. Technically, all the team has to do is move the code to a public repository and start dealing with external bug reports and code contributions the way they have been dealing with such input from other members of their company.

However, a few legal barriers may stand in the way. If programmers incorporated outside code into the project, the legal department must examine all licenses to ensure they have the right to open the code and that their license is compatible with the licenses on the code they incorporated. Some branding review may also be involved. At PayPal, the legal team consulted with open source experts and developed a web-based process for going through the necessary steps.

Modern programmers learn to thrive on change. Aside from new technologies and tools, a change of culture can help them stay nimble and keep skills up to date. InnerSource is a step toward all these achievements.

About the Author

Andy Oram is an editor at O'Reilly Media. An employee of the company since 1992, Andy currently specializes in open source technologies and software engineering. His work for O'Reilly includes the first books ever released by a US publisher on Linux, the 2001 title *Peer-to-Peer*, and the 2007 bestseller *Beautiful Code*.