Practical Guide to Improve Your Open Source Development Impact

Ibrahim Haddad, Ph.D.

VP R&D and Head of Open Source Group

Samsung Research America

September 2017

In this article, Haddad provides a practical guide on improving your open source development impact via ten recommended practices and using the Linux Kernel as a case study. He also offers an overview of the experience with Samsung's open source group and some of the lessons learned throughout this experience.

1. Introduction

Open source development has its own set of challenges, but it becomes easier if you have a clear plan to follow. For instance, the Linux Kernel is the largest collaborative software project in the world, and getting involved in the development process can be a challenge. If you're one of the growing list of companies that relies on the Linux Kernel for their products and services, investing time and money into improving your internal development ability can pay of immensely in the long run.

Fortunately, since so many companies and individuals have been successful at contributing to the Linux Kernel, there is a pretty clear path on to improve your own Kernel contributions and aim for a leadership role. This practical guide will cover a number of practices that enterprises can adopt to help grow their footprint in large open source project using the Linux Kernel project as a case study.

2. Top Recommended Practices

1 Hire key developers and maintainers from the project's community

This critical step allows you to gain skills and recognition. Two or three people are a great start towards making a noticeable impact in a large project such as the Linux kernel, attracting further hires, and allowing enough resources to mentor existing junior developers.

You also need to align corporate interests with individual interests: it's very hard to motivate a senior open source developer to work when their personal interests don't meet with corporate interests in a given project. For example, a Linux memory management expert may not be interested in working on file systems, a corporate priority. Therefore, finding a match in interests is critical for a long lasting relationship.

2.2 Allocate Time for Upstream Contributions

The core principle for hiring open source developers is to support your open source development and upstream activities. There is also the expectation that they should support product teams in their expertise areas. However, it's not uncommon for product teams to exercise their influence in an attempt to hijack the time of the open source developers by having them work on product development as much as possible. If this happens, many open source developers will head to the door, seeking a new job that allows them to work on their upstream project before you realize what just happened.

Therefore, it's important to create and maintain a separation of upstream work and product work. In other words, it's recommended to provide your open source developers with guaranteed time to meet their upstream aspirations and responsibilities, especially if they are a maintainer. For junior developers or other internal developers who are using open source in product components, such interactions with the upstream community will increase their language, communication, and technical skills. In the absence of such an upstream time guarantee, it's easy for these team members to be sucked into becoming an extension of product teams, resulting in their upstream focus drying up in favor of product development.

2.3 Create a Mentorship Program

Setup a mentorship program where senior, experienced open source developers provide mentorship to junior, less experienced developers. Typically, the mentorship

program would run for 3 to 6 months; during this time, the mentor should supervise the work of the mentee, assign tasks, and ensure proper results. The mentor would also do code reviews for anything the mentee produces, and provide feedback before the mentee pushes the code to the upstream project.

The goal is to increase the number of developers the company has contributing code to the upstream project, and to improve individual effectiveness by increasing the quality of code and the percentage of code that is accepted into the upstream project. Generally speaking, no more than 4-5 mentees should be assigned to a given mentor, and ideally they should work in the same area as the mentor to make code reviews more efficient.

2.4 Formalize HR Track for Open Source

Create an open source developer track in your HR system so people hired as open source developers have a good sense of how their career will progress within the company versus other non-open source developers. Additionally, you should adjust performance-based bonuses to include goals related to open source development work. The metrics by which the performance of proprietary or closed source developers are measured are often different than those of open source developers.

Finally, allow a work from home policy for open source developers, regardless of the general corporate policy related to this. Lately, we have witnessed a reverse in work from home policies across companies where many have either banned or created strict limitations to working from home. In the open source world, a work from home policy is almost mandatory because open source experts are located all over the planet, and this policy is often the only way to hire them.

2.5 Training

It's impossible for any company to hire all the senior and most expert developers in a given domain. This concept applies to the Linux Kernel and any other prominent open source project. Therefore, you must have a way for your company to increase the competence of its developers in a given technical domain. In addition to technical training, you'll also need training to teach the open source development model and the basic concepts of open source legal compliance.

Sample training courses include:

- Technical training that covers the various areas in the Linux Kernel. Maintainers or senior developers usually present this to grow internal Kernel expertise; this expertise is vital to pass on given how challenging it is to hire expert Kernel developers.

- Open source development methodology course that teaches staff that's new to open source how open source and Linux Kernel development works and how to best get engaged.
- Open source compliance course that teaches staff the basics of compliance principles and open source licensing. This should also be used to inform them of your company's policy and process. [The Linux Foundations offers a free online open source compliance training for developers. The course is available from here.]

2.6 Participate in Open Source Events

Support your developers to attend and participate in open source conferences and events, including local community meetups, hackathons, and summits. Such participations help them connect at a personal level with their peers, build relationships, have face to face social interactions, and participate in technical discussions that guide the project direction.

If your developers have work that others might be interested in, help these developers prepare content to present. Finally, you can also sponsor events, both big and small, to increase external visibility within the project's community. These events are also a great venue to look for talent!

2.7 Provide a Flexible IT Infrastructure

Provide a flexible IT infrastructure that allows open source developers to communicate and work with the open source and Linux Kernel community without any challenges. Additionally, setup internal IT infrastructure that matches the tools used externally to help bridge the gap between internal teams and the Kernel community or any other open source project community for that purpose.

There are three primary domains of IT services that are used in open source development: knowledge sharing (wikis, collaborative editing platforms, and public websites), communication and problem solving (mailing lists, forums, and real-time chat), and code development and distribution (code repositories and bug tracking). Some or all of these tools will need to be made available internally to properly support open source development. There is a chance this might conflict with existing company-wide IT policies. If so, it's vital to resolve these conflicts and allow open source developers to use the tools they are familiar with.

2.8 Track Developer Code Contributions

Create internal system to keep track of developer contributions and impact. Contributions can include upstream development, supporting product teams,

knowledge transfer (mentoring, training), visibility (publications, talks), launching new open source projects, and establishing internal collaboration projects with other teams or groups.

There are several toolkits that help track source code contributions; for instance, The Linux Foundation uses a tool called gitdm, which produces the data reported in the Linux Foundation yearly Linux Kernel report. This can be used to track both individual developers as well as the overall team performance. Individual developers can be tracked for the number of patches they submit, the patch acceptance rate (patches submitted divided by patches accepted), and the type of patch (e.g. if it is a new feature, enhancement of existing functionality, bug fix, documentation, etc.).

Other tools like GrimoireLab can also be used to chart and visualize the metrics you want to track. One of the most basic things that can be tracked is the number of commits and lines changed over a specific period of time, such as every week, month, or year.

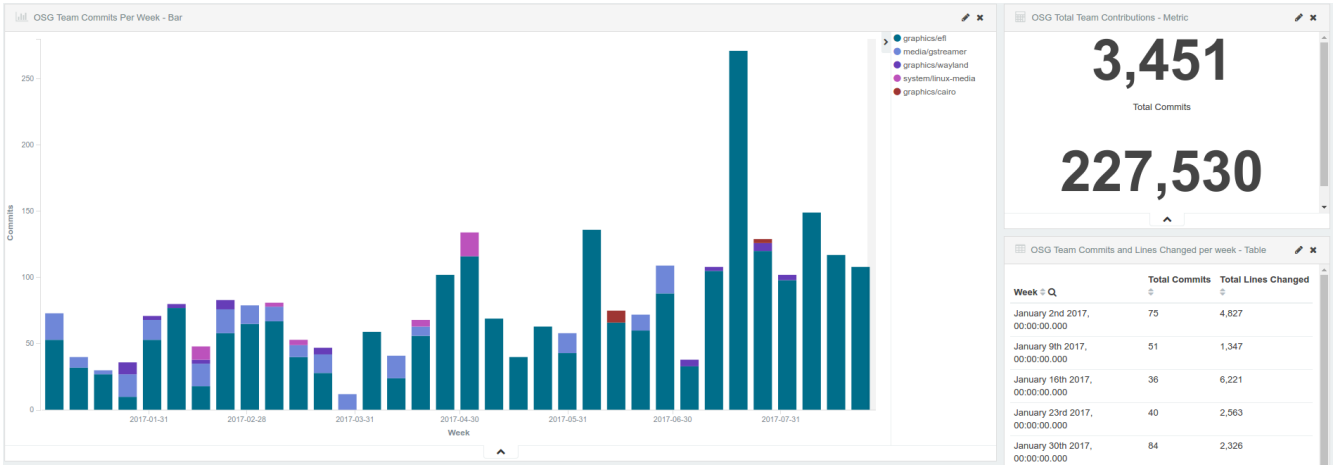

Figure 1: The total commits and lines changed per project per week is a good place to start tracking metrics.

With this data, you can compare contributions from various internal development teams to identify where source code contributions are coming from. Charts that compare various internal teams for their cumulative contributions, percent of total contributions, and the amount of time it takes to get code committed upstream can be useful for this (Figures 2 and 3).

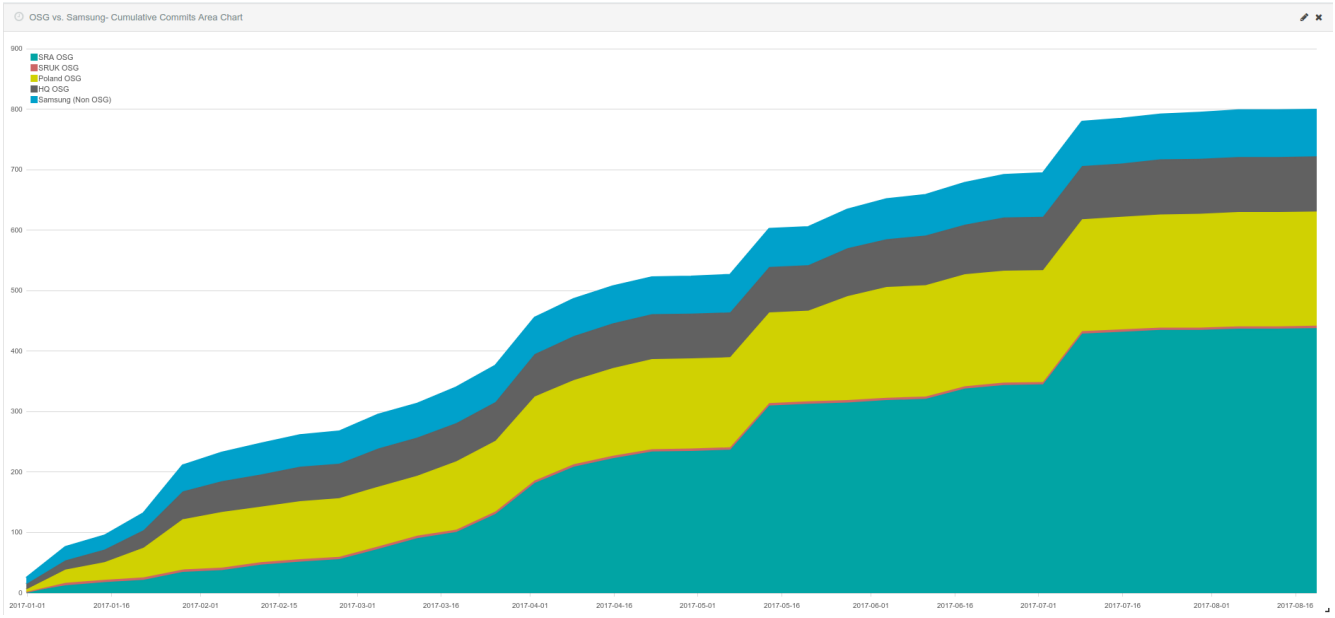

Figure 2: Cumulative contributions over time can be tracked to compare internal teams, and identify teams that are increasing their involvement in the kernel community.

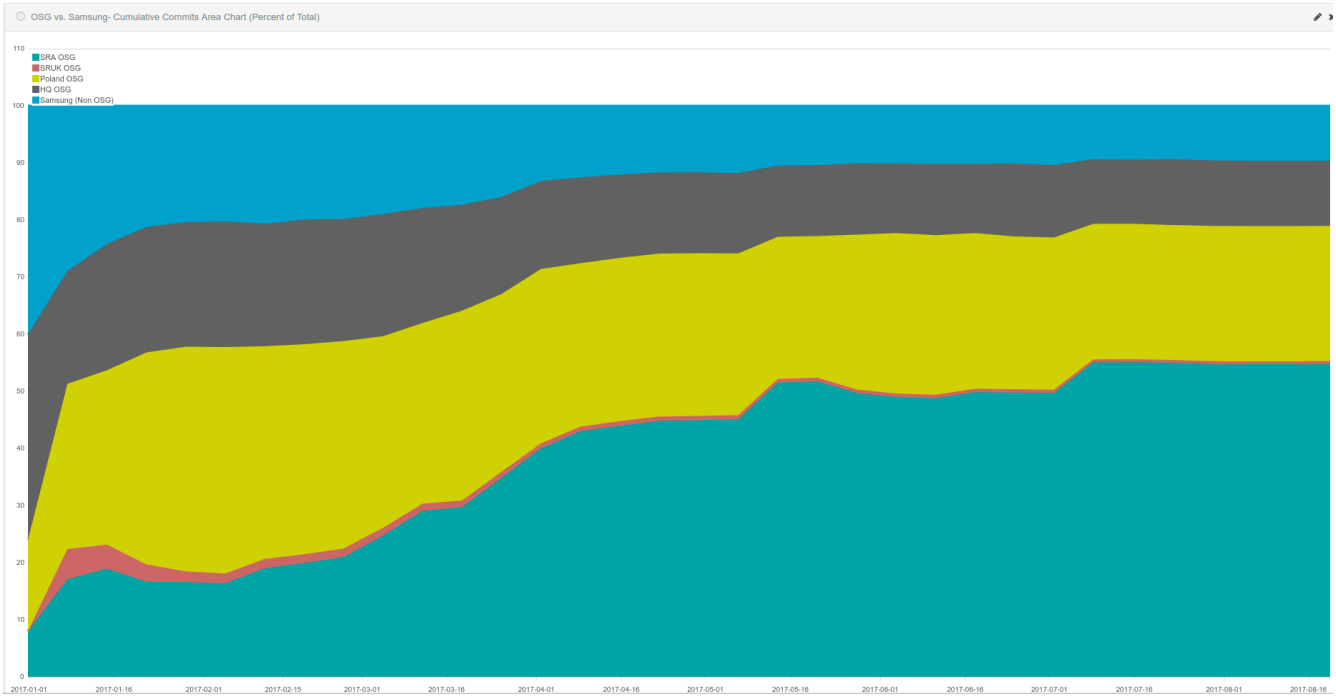

Figure 3: Displaying your company's contributions as a percent of total over time allows you to identify the teams that contribute the most code.

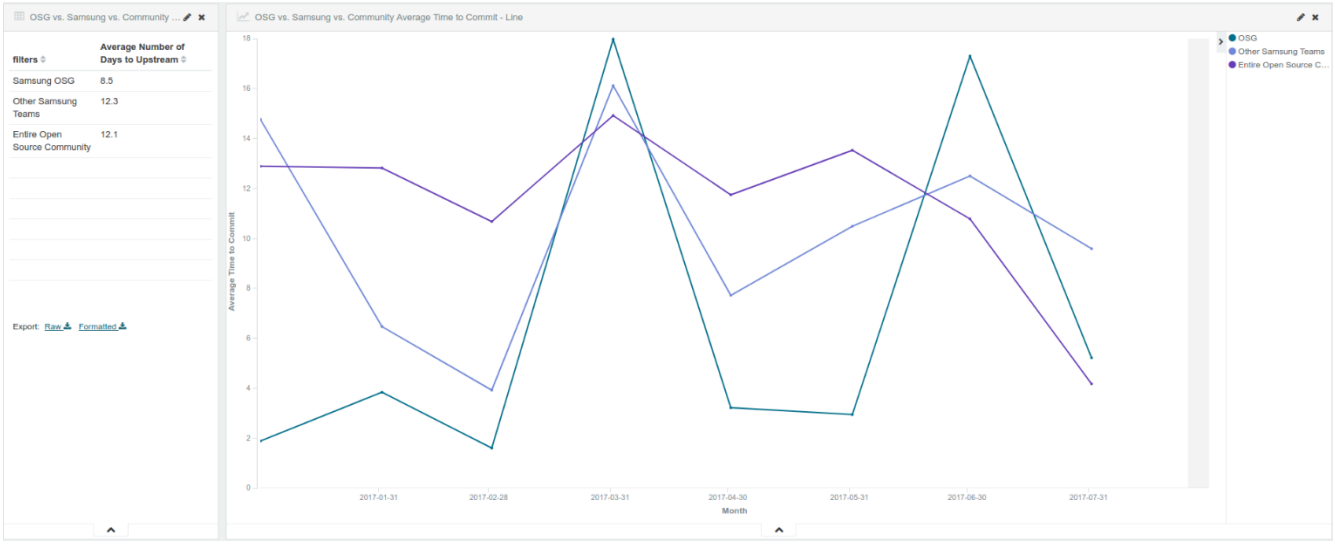

Figure 4: The amount of time it takes to commit code upstream can be valuable for tracking your development efficiency. This table and chart shows how quickly various teams are getting their code contributed upstream and compares it to the community as a whole.

You can also use these metrics to compare your performance to other companies who are involved in the Kernel ecosystem for instance (Figure 5). This helps you be better informed about the overall developer ecosystem for the project.

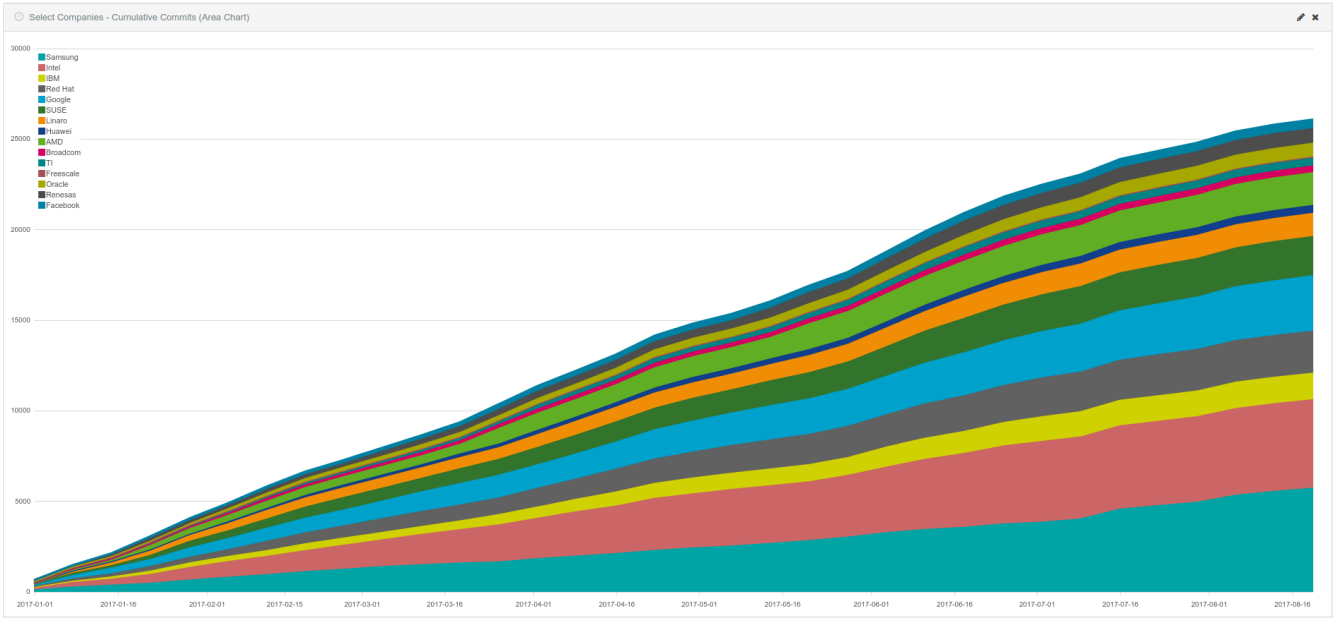

Figure 5: Cumulative contributions can be sorted by company to see how your company stacks up against others.

These metrics provide a much better idea of where your strengths and weaknesses are and can help inform your overall development strategy.

2.9 Identify Focus Areas with Broad Impact

Contribute to and focus on areas that benefit more than one business unit or more than one product. This allows you to provide value and show ROI across multiple business units and increases your chances for more funding and support.

2.10 Foster Internal Collaboration

Create collaboration projects with other business units that use the specific open source projects in their products. These collaborations can take one or more of many forms:

- Deliver training to their developers.
- Run a workshop on a specific topic or problem.
- Develop new functionality.
- Troubleshoot and resolve issues and bugs.
- Upstream existing code for which they have no resources to do.
- Help get them off an old fork and on to a mainline version.
- Etc.

The goal of these collaborations is to help the products teams understand their needs and fulfill their product goals via open source enablement.

3. Remarks on Samsung's OSG Experience

3.1 Brief Overview

Samsung's Open Source Group (OSG) was established in February 2013 to support two primary functions: the first is to provide open source leadership within Samsung by helping other divisions in the company understand how to participate in and benefit from open source development. The second is to serve as Samsung's representatives in the wider open source community. The mandate of the team is to focus on enhancing key open source projects and technologies via active contributions to them, and to be actively involved and engaged with various open source organizations and foundations.

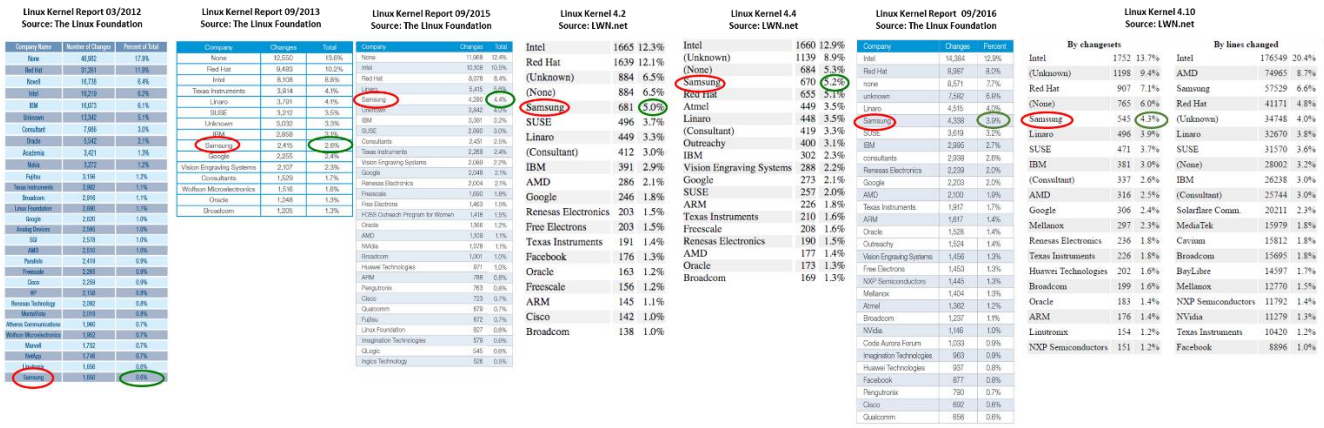

Figure 7: We also track our contributors via the Linux Kernel and LWN's kernel reports.

As you can imagine, the Linux kernel is of strategic importance so we chose it to be a focus area among many other projects. Figure 7 provides an illustration of our progress as a company to become a top contributor to the Linux Kernel; we are now regularly among the top 5 contributors by changesets. The same progress applies to several open source projects that we deemed critical to the development of our products.

Our goals are to reduce the amount of work needed from product teams, minimize the cost to maintain source code and internal software branches, improve code quality, produce faster development cycles, produce more stable code to serve as the base for products, and improve company reputation in key open source communities. As such, the group has had both direct and indirect positive impacts on Samsung products.

3.2 Direct Enablement

Our team has had a direct impact on Samsung in a few ways. For starters, our open source engineering team has been effective at fulfilling open source development requests from R&D and product teams. We've also helped bring internal Samsung code into various open source projects. Finally, we've implemented numerous drivers related to Samsung products into upstream code. Our expertise in committing code upstream has made us particularly valuable with these components, and our efforts have resulted in a reduction in the amount of effort required to maintain kernel code (as an example) that's used in Samsung products and services.

3.3 Indirect Enablement

Our impact goes beyond the code we contribute to the various open source projects. One major area we make indirect contributions is by influencing the various

communities through participation in technical discussions. Internally, we participate in policy and architecture discussions and decisions to ensure our internal decisions match the direction of the specific project community.

We also provide assistance to resolve compliance issues and support our compliance team with the open source compliance inquiries they receive. Finally, we also help stabilize the code of the various open source projects we're involved in, improving its overall value for all people that use it, including Samsung.

3.4 Challenges

We can group the challenges we've faced under 3 categories: culture, processes, and tools. Within each of these categories, there are several elements that you will need to address to fit the open source model.

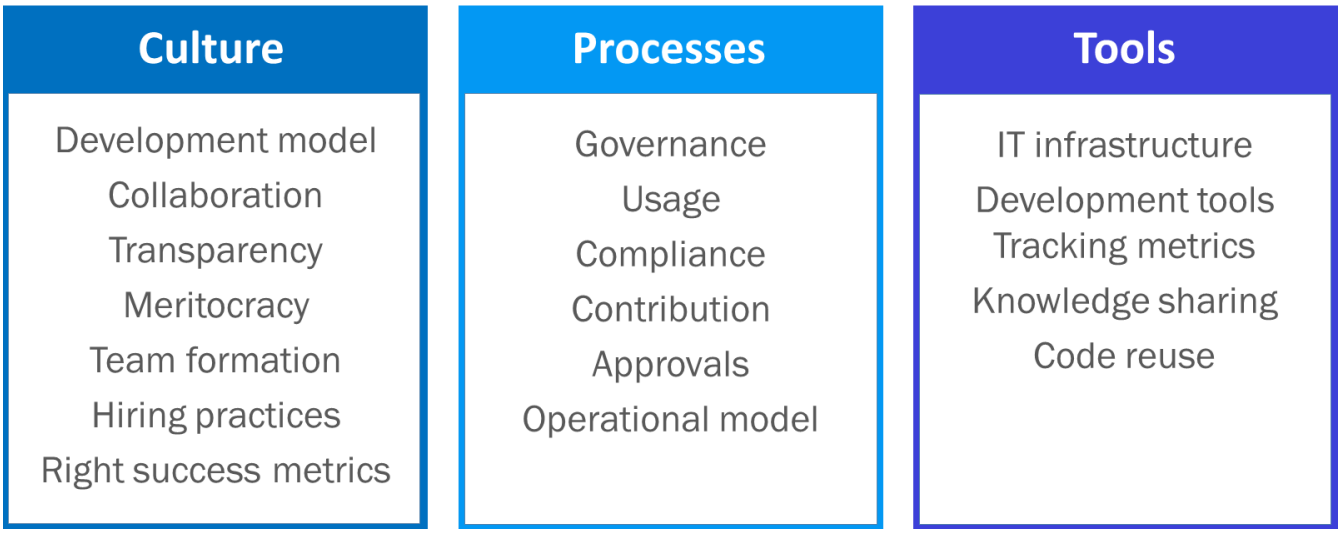

Figure 8: Challenges open source dedicated teams face in an enterprise setting

Our cultural challenges have stemmed from the fact that there is a gap between traditional software development practices and the requirements of open source development. We've bridged this gap by hiring open source experts and having them train other groups that aren't familiar with the open source development model. We also provide open source engineers with adequate time to fulfill upstream responsibilities so they can provide adequate open source leadership for Samsung.

Open source development is dynamic, moves very quickly, and has special requirements for compliance, companies that don't adapt their internal processes to meet this type of development can easily get left behind. Developers need to be enabled to contribute

code upstream quickly and any internal code policies need to be modified to allow this. Additionally, it's vital to have a team in charge of maintaining proper open source compliance to avoid costly legal problems.

When we started the OSG at Samsung, many of the tools Samsung used weren't compatible with the open source development model. We've made major efforts to create a setup that fulfills the needs of the OSG and meets corporate IT guidelines. For instance, today, we issue Linux devices that works with all of the tools our engineers need to participate in open source development. This environment allows developers to join our team without requiring any major change to the way they work. We also support work from home and in fact we only have two developers in our Silicon Valley offices and all other staff are remote.

3.5 Lessons Learned

Given the 4+ years of experience we've had since the launch of Samsung's Open Source Group early 2013, here are some of the lessons we have learned align the way:

1. It takes considerable time to grow internal open source expertise. Our goal is to find people who have enough peer recognition to be influential in the community. There are typically 3 pillars to this: domain expertise, open source methodology, and working practices.
2. Internal company dynamics need to be favorable to open source efforts. Implementing these practices requires a shift from traditional software development practices to a more open and collaborative mindset. As an open source leader inside your organization you will face several challenges in terms of funding resources, justifying ROI, getting upstream focus, etc. These often require a major shift in mindset and a lot of education up the chain.
3. These open source practices typically require an IT infrastructure that is free from many standard, limiting IT policies. It took us years of constant discussion and negotiation to break from the traditional IT setup into a more flexible environment that supports our open source development. We made it work for us and with enough persistence you also can make it work for your open source team.
4. Proper open source metrics are required to drive the desired development behavior. The traditional metrics often used in product organizations don't apply in the context of open source development. For example, we've had multiple instances of desired functionality being implemented upstream because of OSG developers that lobby support from the community. In this case, the number of

changesets or lines of code doesn't matter nearly as much as the technical leadership team members provide to get code upstream and reduce our downstream maintenance efforts. The metrics we track account for things like this.

5. The company needs to have a simple internal approval model for open source contributions. Throughout the years, Samsung has moved from highly complex and cumbersome policies to a more simple approach for receiving, reviewing, and approving source code contributions. It's a function of balance between all parties involved: legal, engineering, and open source. The compromise we have now supports the dedicated open source team which has a blanket approval to contribute to a number of open source projects. This is not the case for other teams who need to get different levels of approval depending on the nature of the code being contributed (e.g. simple bug fixes, code to improve existing functionality, code that offers new functionality, or starting a new project).
6. The company must share information and priorities across different divisions. To illustrate this, assume you are in an open source team and you are requested to support the implementation of a driver, but you are unable to get access to the hardware manual and instructions. This sounds a bit like playing darts with the lights off, and it is. Information sharing is a critical component to successful internal collaborations between the open source teams and everyone else.
7. Focus your contributions on upstream projects that would directly benefit the company's strategy and products. In open source development, it's very easy to get carried away hopping between different interesting projects. In an enterprise setting where the open source group is considered a cost center, your driving force should be to focus on open source projects that supports product development. In our case, we do a yearly review of the product portfolio in an effort to have our involvement be in open source projects that are commonly used across as many products as possible. This list is then prioritized based on several factors, and we focus our efforts on the top projects. A methodology that drives your priorities is a great way to stick to what's important, justifiable, and fundable.
8. Be the upstream partner for product teams; they often feel like being inside a pressure cooker, especially in a consumer electronics environment. They often seem understaffed, lack critical resources to support parallel upstream development, and are under constant pressure for feature delivery within tight schedules. In such an environment, it's very easy to overlook the benefit of upstreaming in favor of short term time savings that can unfortunately lead to technical debt that has a higher cost in the long term. Open source teams can

help by being a partner that focuses on delivering important code upstream, reducing this technical debt.

9. Grow open source talent in specific technology areas relevant to your products. It's easy to hire a few resources from outside the company, but there are several limitations to this approach. The alternative approach is to convert your existing developers into open source contributors via training on the technical domain and open source methodology. These developers can then be paired with a mentor to further expand their skills.
10. Encourage developers outside the open source team to learn from and contribute to the open source community. We provide as much help as we can with upstream code contributions, but our resources are limited, and we don't always have the deep understanding of products that might be needed to adequately identify where code can be upstreamed. Better involvement in the open source community from teams outside our own allows us to get more important code upstream, and improves our ability to interact with the community.

4. Closing

Open source leadership can’t be given, it must be earned. This leadership is earned through constant participation and contribution. While it can't be taken away, it can be lost through a lack of participation. The only way to ensure your company maintains open source leadership is through regular, ongoing participation and contribution.

Hopefully this article hasn't made the task seem too daunting, but if you follow some of the practices provided here, it will go a very long way towards developing the internal open source expertise you require. You can then leverage that expertise to improve your products and services while reducing code maintenance costs. Samsung has had considerable success through these strategies, and if you follow them, so can your company.

Happy hacking!

References

Samsung Open Source Group
https://blogs.s-osg.org/

GrimoireLab – An open source software development and community analytics platform.
https://grimoirelab.github.io/

Compliance Basics for Developers – A free online training course offered by The Linux Foundation.
https://training.linuxfoundation.org/

gitdm – The "git data miner" is a tool created by Jonathan Corbet (LWN.net) and Greg Kroah-Hartman (Linux Kernel Maintainer and Fellow at the Linux Foundation) to track Linux kernel patches.
git://git.lwn.net/gitdm.git

Facade – A tool for monitoring who is contributing code to git repositories. It was authored by Brian Warner (Manager of Engineering and Strategy for Samsung's OSG).
https://github.com/brianwarner/facade

Acknowledgments

The author would like to express his sincere gratitude to Ben Lloyd Pearson (Operations and Strategy, Samsung Open Source Group) for the continuous support, reviews and edits (following the open source model) that shaped this article and helped get it to the finish line. Big thanks also goes to Brian Warner (Manager of Open Source Engineering and Strategy, Samsung Open Source Group) for his review and feedback.

About the author

Ibrahim Haddad (Ph.D.) is Vice President of R&D and the Head of the Open Source Group at Samsung Research America. He is responsible for overseeing Samsung's open source strategy and execution, internal and external collaborative R&D projects, supporting M&A activities, and representing Samsung in open source foundations. He is currently serving as Vice President of the Open Connectivity Foundation and the Director on the Board representing Samsung Electronics.

Twitter: @IbrahimAtLinux