Openness Index – Determining the True Openness of Open Source Projects

Ibrahim Haddad, Ph.D.

The motivation for writing this article originated from various discussions evolving around what makes a project be a true open source project. Different people have different opinions and thoughts about the various metrics by which we can measure or sense that trueness. In this article, we explore such metrics that together can help define the true openness of a given project and conclude with some best and worst practices in an open source project touching a dozen different areas.

CONTENTS

OPENNESS INDICATORS

Insert intro

The success of an open source project depends on many metrics, most of which can be grouped under a single category: openness.

- License
- Governance
- Access
- Processes
- Development
- Community Structure
- Tools
- Release Openness
- Roadmap
- Diversity of Contributors
- Compliance Considerations
- Documentation

In this article, we examine each of these indicators, and discuss how their openness can contribute to the health and openness of the overall project.

LICENSE

The license of an open source project determines the rights to use, copy, modify, and distribute the code. In some cases, the license might require a specific copyright assignment, copyright license, or patent grant. The choice of license for an open source project is an essential factor in determining the openness of the project. Ideally, open source projects should only use Open Source Initiative (OSI) approved licenses. Custom licenses are a clear sign that a project is not truly open because it creates potential legal liabilities that can result in major risks for participants and distributors.

Developers should be able to create and distribute derivatives of the source code for their own projects or to re-use the code in other projects. To allow this, the project needs to be available under an appropriate OSI-approved license that provides these freedoms.

GOVERNANCE

Governance determines who has influence and control over the project or platform beyond what is legally required in the open source license. A project's governance model addresses difficult questions such as:

- Who makes decisions for code inclusion and releases, and how?

- Who owns the copyright on contributed code?
- Who can be the lead maintainer or architect?
- How can contributors become maintainers or committers?
- How can the project raise money and who decided on how this money is spent?
- Who decides the project's direction?
- Who decides on the project's roadmap?
- Should the project have a Technical Steering Committee or Compliance and Certification Committee? Who can be on them?
- What are the conditions under which the project permits the use of its trademark?
- The openness/transparency of the project.
- Who can participate in the discussions and decide on critical matters?
- How transparent are the decision-making processes?
- Can anyone follow the discussions and meetings that take place in the project?
- Can anyone create derivatives based on the project?
- What compliance requirements are there for creating derivative software? How can the project enforce these requirements?

Open source projects with an open and transparent governance model have better chances to grow, have a healthy environment, and attract developers and adoptees.

ACCESS

A key indication of project openness is how accessible the project's resources, communications, and history is to external participants. For starters, an open project will provide the same level of availability of source code to all developers, meaning there is a complete lack of any favoritism to developers via priority access.

Additionally, open projects provide access to developer tools such as mailing lists, forums, bug-tracking systems, source code repositories, and documentation. Participants are able to join discussion platforms, decision-making mechanisms, and project roadmaps so it is possible to understand why and how the project makes decisions.

PROCESSES

A project with a high degree of openness will have clearly defined processes for how things work in the community and how to contribute to the project. For starters, a clear development process should outline how to incorporate code into the project, the release process and schedule of the project, and any requirements developers need to meet to get their code accepted. This should also include guidelines for participation that demonstrate community best practices for things like patch submissions and signing-off on code contributions. Finally, there also need to be processes to guide users who want to contribute bug reports or issue requests as well as processes for developers to resolve them.

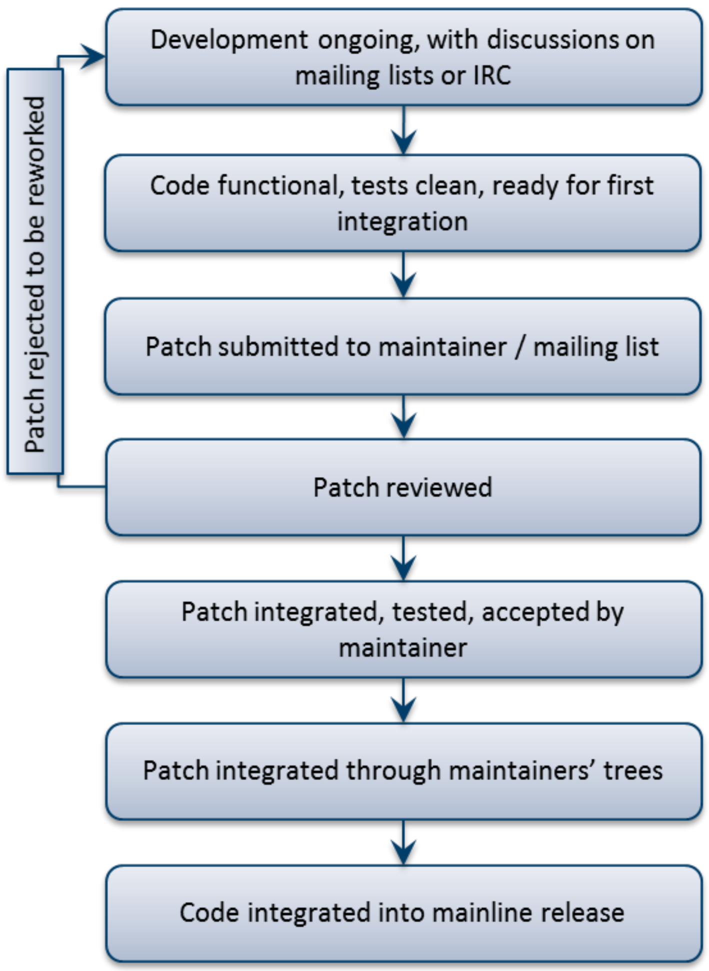
Development ongoing, with discussions on mailing lists or IRC
Code functional, tests clean, ready for first integration
Patch submitted to maintainer / mailing list
Patch reviewed
Patch integrated, tested, accepted by maintainer
Patch integrated through maintainers' trees
Code integrated into mainline release
Patch rejected to be reworked

Figure 1: Sample code submission process. Open source projects are highly encouraged to document their development processes. The illustrated process assumes a small project. In complex projects, patches proceeds through several layers of maintainers.

DEVELOPMENT

An open development process enables developers to influence the content and direction of the project via contributions. This requires a transparent contribution and acceptance process that provides clear feedback on updates to contributions as they are incorporated into the project. This transparency should also extend to enable external participants to identify the source from where code contributions originated.

Release early, release often is a practice that has been integral to open source software for most of its history. This is one of the primary practices that allows open source communities to innovate at a rapid pace with a high quality of code because it creates a much faster feedback loop between developers, testers, and users. Releasing early allows feedback at an earlier stage of development so new ideas can be incorporated while the code is still flexible; it also allows any potential issues to be flagged more quickly. Releasing often results in smaller changes that are easier to understand, debug, and improve which makes it much easier to maintain a rapid development pace.

COMMUNITY STRUCTURE

Open source project communities usually start with a flat structure and transition to a hierarchical structure as they grow in terms of contributors and the body of code becomes more complex requiring additional maintainers. From that perspective, the code leadership evolves around committers, maintainers, and reviewers (please note that not all projects support these levels of distinguished contributors). A committer is any individual who can commit code into the mainline project, maintainer and reviewer are alternative versions of this position and the terms are often used interchangeably.

A simplest indicator for how open the community structure of a project is around two key factors:

1. Meritocracy: The concept that individuals responsible for the project leaderships got their roles based on talent, effort and achievements in the project.
2. Accessibility: A key component of community openness is the accessibility to becoming a trusted committer. This process should be clearly documented and equitable so that any contributors have the potential to become a committer.

In some cases, open source projects have a number of enterprise sponsors and the project is formalized around its own foundation with membership fees that offer a number of rights, privileges and responsibilities for the sponsoring enterprises. A final point to consider is whether the community of such projects offer an equal and democratic environment for contributors who do not come from a sponsoring organization and whether they have equal chances in being promoted to trusted contributors or maintainers.

SMALL SIZE PROJECT

Small open source projects tend to have only one maintainer overseeing the contributions from several developers into a single body of code. Releases are often made available when the developer community feels that the release is ready versus having a specific target date with a list of features that need to be locked by that date. Such projects have a flat hierarchy and sometimes committer status is awarded to certain individual developers who demonstrated consistent level of high quality code and thought leadership.

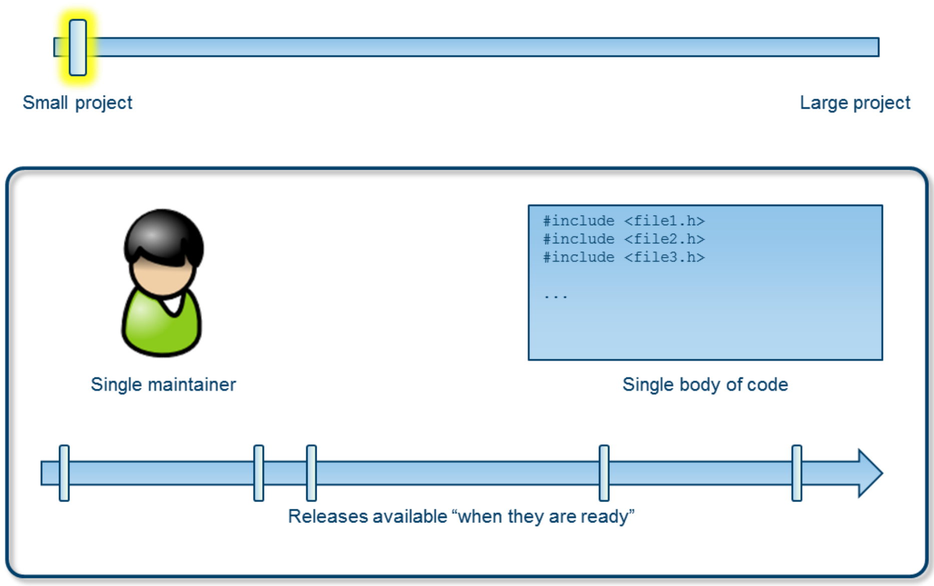

Figure 2: Sample structure of a small project

Medium size project

As the project grows, the body of code becomes larger to handle by a single maintainer, as well as the number of contributors sending their patches. At this point, the project starts to take a modular form in terms of code partitioning into sub-system with a top maintainer and various possible committers entrusted to oversee the various modules. A set release cadence becomes necessary and the criteria to move from a development branch into a release candidate and then a release becomes more stringent.

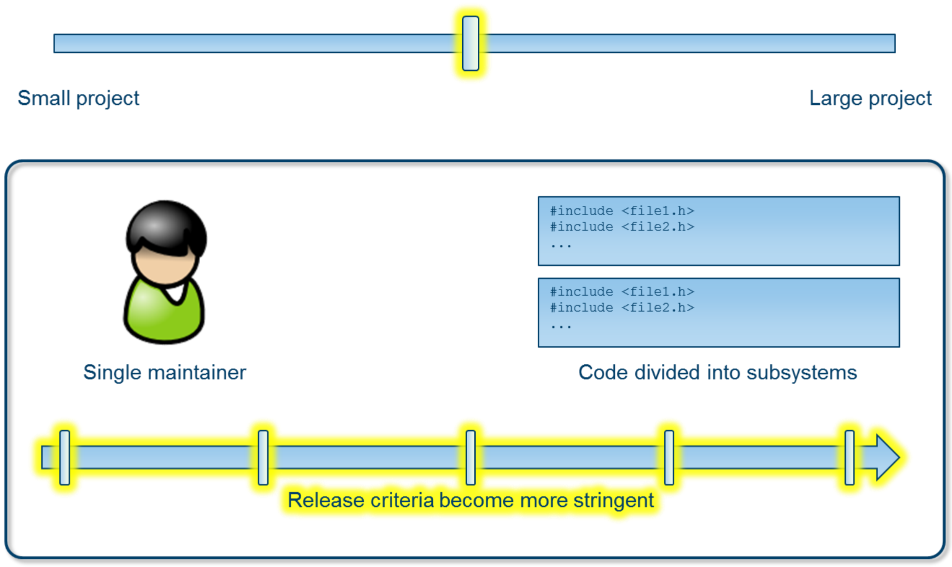

Figure 3: Sample structure of a medium size project

LARGE PROJECT

Large open source projects are often hierarchically structured based on the various source code modules with multiple sub-maintainers, and in some cases sub-sub-maintainers. The code base is very large to be managed by a single of even just a few maintainers. Such large projects follow a rhythmic release cadence and different developers acting as dedicated release managers.

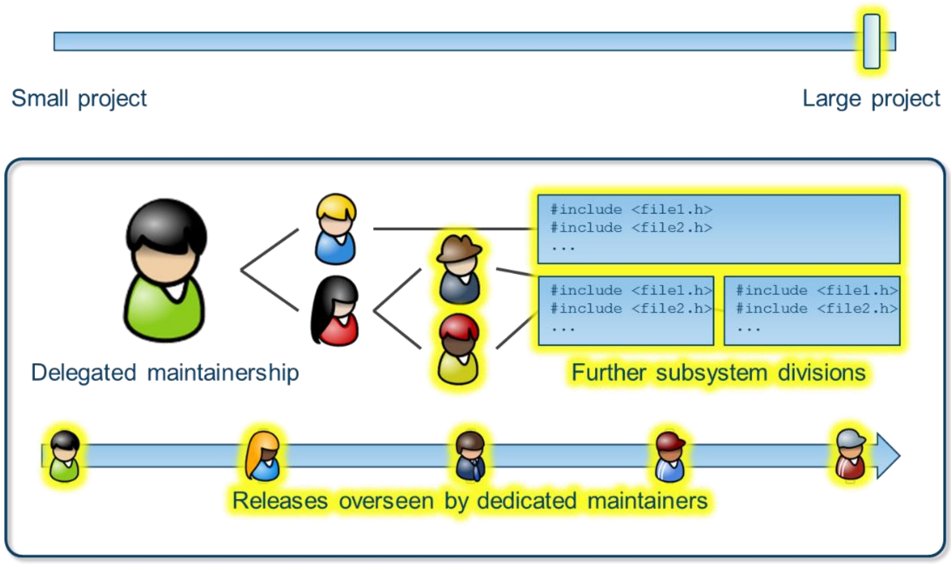

Figure 4: Sample structure of a large project

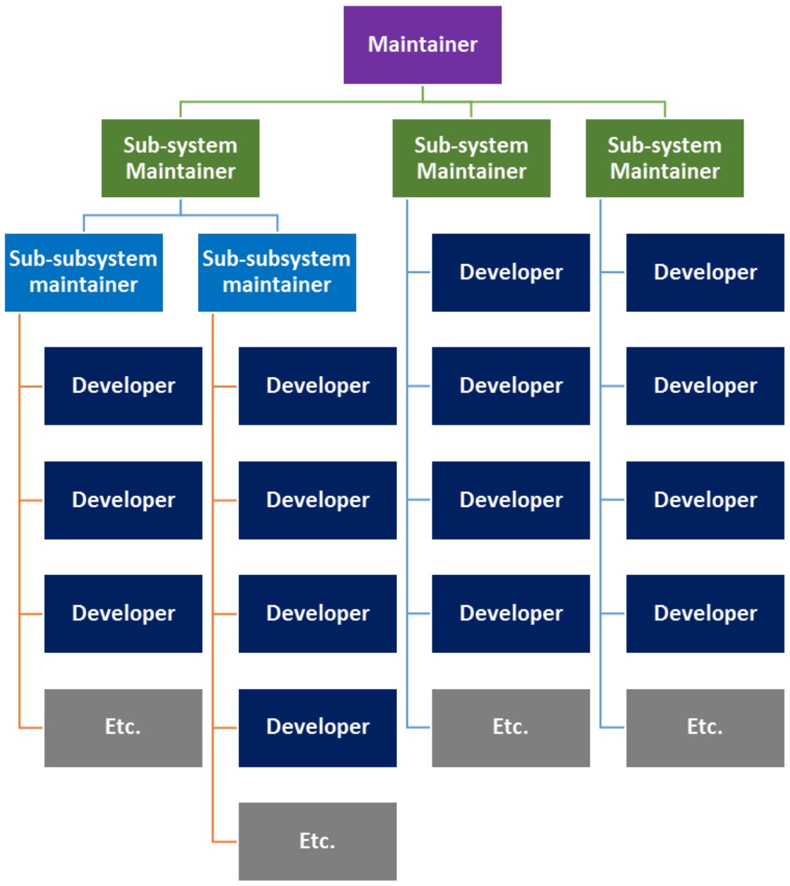

Figure 5: Large project with a hierarchical community structure

The Linux Kernel is a great example of a very large project with hierarchical community structure similar to the structure illustrated in Figure 5.

CHARACTERISTICS OF A GREAT OPEN SOURCE COMMUNITY

Great open source communities may differ in what they work on and how they implement the structure and processes of their projects but they share several characteristics:

- Community members work together for a common goal with a high sense of cooperation.
- Project participants feel free to expression their opinions, share their ideas and discuss with other project members.
- Community members chose their maintainers and committers based on their expertise, level of contributions, and thought leadership. The community maintains a clear process for the selection criteria.
- The project's community is accessible to newcomers as users of the project or developers who wish to participate and contribute. Open development strives for inclusiveness.
- Great open source communities are very transparent with strong emphasis on open decision-making processes and communication.

- Great open source communities are resilient to organizational change. Leadership is earned with experience and with approval and consensus from community members. If individuals cease to participate in the project, there are others to take their role with minimal disruptions to the project and a clear process to guide the selection of the new leaders (maintainers).
- Great open source communities work to ensure that those who fall in minority populations are not treated differently. These communities give a voice to minority populations through frequent consultation with members of those societies about how the community can improve to meet their needs better.
- Great open source communities foster a feeling of connection and collaboration among its members by providing plenty of opportunities for interaction. Creating a feeling of connection makes members more motivated working towards the projects' goals.
- A healthy and strong open source community is also a diverse community. Diversity can be at different levels. For instance, the diversity of the contributor base where contributors comes from academia, corporates or as simply hobbyists. Other diversity metrics include sex and ethnic backgrounds. Some open source projects are doing fantastic efforts to increase the diversity of their contributors and to encourage new participation. For example, OpenStack

COMMUNICATION TOOLS

Collaboration in most open source communities is centered on a relatively standardized set of tools. Tools like wikis, IRC, and mailing lists allow members of the community to communicate with each other. Open source communities often rely on things like Bugzilla, git, and file servers to collaborate on code development, and blogs are often used to inform people about community efforts to keep everyone on the same page. Internal policies and infrastructure must be put into place to ensure your developers can adequately interact with open source communities using these tools.

RELEASE OPENNESS

Until the writing of this article, I did not think of such a thing as "release openness". However, two key release related indicators play a role in highlighting the openness of the project. These indicators are the release goals and cadence, and release notes.

- Release Goals and Cadence: This indicator is very simple at face and we can present it as a series of questions to which the answers should be "Yes".
    - Does the project have a release cadence set and agreed by project contributors?
    - Are there development milestones set for each release?
    - Are these milestones (per release) published on the project's web site or wiki?
    - Is development prioritized based on the cadence and goals of each release?

- Release Notes: Release notes are equally important as release cadence and deserves to be its own headline. In open source projects, with hundreds and possibly thousands of developers, documenting releases is really a fundamental requirement than a feature and should be considered as part of the development and release cycle. Like release cadence, there are many advantages that result from providing detailed release notes such as providing visibility into the project’s progress, documenting continuous improvements to the projects with every release, providing a great reference for new users of developers joining the project, and in general using it as a communication tool.

The availability of the release cadence, goals and notes add having the ability to discuss these in an open fashion adds a great dimension of transparency to the project and makes all participants feel ownership of the release.

ROADMAP

At a high level, roadmaps provide high-level overview of the project’s goals and deliverables for that release. Open source projects that maintain a roadmap achieve several advantages and are able to:

- Communicate the plans and goals for each release (minor, major, etc.),
- Manage the expectations of its users and developers by generating a shared understanding across everyone involved in the project, and
- Expose the project’s plans to other open source projects that possibly rely or use them as a dependency.

INTELLECTUAL PROPERTY CONSIDERATIONS

Although not quite an openness indicator about the project, compliance, copyright and other IPR concerns, are openness indicators about the contributors.

Open source projects deal with these concerns differently. Some projects adopted a sign-off-by process, others require a CLA, and the majority of projects rely on common trust with their contributors.

The signed-off-by process ensures that every single line of code accepted into a project has a clear audit trail. It is a developer’s certification that they have the right to submit code for inclusion into the project, and that they agree to the Certificate of Origin. The Linux kernel community was the first community to implement and run a signed-off-by process. That requires all contributors to sign-off their code, which indicates the contributor certifies the code as outlined in the Linux Kernel Developer Certificate of Origin. That signature communicates that the contributor has created or received the patch in question under an appropriate open source license that allows it to be incorporated into the project’s code base. It is used primarily to establish a chain of people who take responsibility for the copyright status of the code in the contributions and to avoid the incorporation of copyrighted code that is not covered under an open source license.

Other projects require either developers or their employers signing a Contributor License Agreement (CLA). The purpose of a CLA is to ensure that the guardian of a project's outputs has the necessary ownership or grants of rights over all contributions to allow them to distribute under the chosen license. In some cases, this even means that the contributor will assign the copyright in all contributions to the project owner; in other cases, they will grant an irrevocable license, which allows the project maintainer to use the contribution.

DOCUMENTATION

An open source project can provide different types of documentation to help its community of both users and developers. Historically, documentation has been an area that is lacking and requires improvements. However, this is changing and many of the projects, especially those hosted within an open source foundation, have great documentation that cover all areas of the projects. In the following sub-sections, we examine three core areas where documentation is essential.

- Project
    - Mission
    - Governance
    - Community structure
    - Release cadence
    - Roadmap and priorities
    - Use cases
    - FAQs
    - etc.
- Documentation targeted for users:
    - User guide and tutorials
    - API guide
    - Architecture overview
    - Installation guide
    - Etc.
- Documentation targeted for developers:
    - Detailed architecture and mapping to code sub-systems/services when applicable
    - Development process
    - How to get involved
    - Guidelines for participation
    - Feature request process
    - Patch submission process
    - Sign-off-by process, when applicable
    - Developer guides and tutorials
    - API guide
    - Etc.

BEST AND WORST PRACTICES

In this section, we present some of the best and worst practices for the various openness indicators we discussed in the previous section.

	Best Practice	**Worst Practice**
License	OSI-approved open source license.	• No license. • Unclear or conflicting licensing terms. • Vanity license.

		• Duplicative open source license with customized language.
Governance	A governance model that gives equal footing to all current and future contributors to the project.	• No governance. • Biased governance that is dominated a given party, usually the founder of the project.
Access	• Project resources are accessible to any users or developers who are interested in the project. • Anyone may participate in the project. • Any participant can earn committer rights by way of code contribution and building community trust.	• Limited access based on seniority, sponsorship level or other factors.
Processes	• Documented processes for requesting a feature, reporting bugs, submitting code, etc. • Code is only committed through the open source process, and never without peer review. • The process to become a committer / maintainer / architect is enforced by the TSC for consistency. • Committer rights in one sub-system do not automatically convey committer rights in another. • Project's community revise its processes frequently or based on incoming feedback to ensure they continue to meet the project's needs at it grows and scales.	• Ad-hoc processes. • Poorly designed processes. • Processes that keep changing or are stale and need improvements to scale and accommodate the development status of the project. • Processes that are not followed or respected.
Development	• Responsibility for development allocated to the individuals with the best capacity to deliver. • The project enforces quality standards when merging code regardless of where the code is coming from. • The project implements multiple levels of review before entering final release. • Peer review is mandatory and public.	• Peer review is not enforced. • Pedigree of incoming code is not verified. • Contributor don't follow sign-off process and still code gets accepted.

Community	• Accessible to newcomers - open development generally strives for inclusiveness. • Focused on visibility with emphasis on open decision-making processes and communication. • Self-organizing where individuals contribute in their areas of interests, or those of their employers. • Resilient to organizational change given that leadership earned with experience. If individuals cease to participate, there are others to take the place.	• Little or no help or support available to new developers entering the project in term of guidance, documentation and mentorship. • Obscure decision-making process.
Community Structure	• Meritocracy drives advancement and acceptance. Contributors who provide the most value to the community are granted project leadership roles. The committers periodically elect a maintainer, who is the ultimate decision maker in the workgroup. • The project welcomes newcomers who have freedom and access to participate in public discussions, development, and testing. • The hierarchy is scalable by consisting of maintainers who oversee specific bodies of code in levels that can be added or removed as needed based on the size of the community. • Anyone can submit patches, and both developers and users are involved in the testing process. The roles of developer and user are closely integrated in open source development, allowing users to have a more direct path to influencing the project.	• Structure biased towards a certain company, coalition or commercial interests. • No clear path for developers on how to they are promoted to a committer, reviewer or maintainer.
Releases	• To protect certain users from the instability of rapidly developing software, projects provide stable releases that restrict the addition of experimental features to	• Unclear structure of releases and branches. • Undocumented. • Uncommunicated.

	provide a reliable version that better supports use cases that rely on stability. • Weekly or monthly stable releases provide users and developers with the newest functionality after it has been tested • Long-term stable versions extend to longer periods and often only include security patches and bug fixes.	
Release Cadence	• The project has a defined cadence for its releases with set goals per release. • The release cadence and the goals to be met by each release are known to all projects stakeholders (users and developers).	• No release cadence – releases happen when community feels the release is ready. • Cadence is not suitable or does not meet the needs of the end users.
Architecture	• Modular architecture allowing it to be more fluid and making it easier for individuals to understand how the various components work and contribute improvements more rapidly. • Less contention over common code: Smaller core with features implemented as plugins or modules reduces collisions. Modularity creates a natural separation of tasks and scope. • Features are available more quickly: Modular designs often have fewer interdependencies and features are released into the development tree as they are completed.	• Lack of vision for a scalable and modular architecture. • Conflicting opinions without a clear path for the future.
Derivatives	Open source license provides the freedoms to create and distribute derivatives.	Non-OSI approved license that limits these freedoms.
Communication tools	Such tools include mailing lists, IRC, are available, and open to anyone wishing to participate in the project.	• Restricted access to some of the communication tools. • Discussions happening in private chat rooms or private mailing lists.
Transparency	Open source communities must be as transparent as possible to attract new participation. • Contribution transparency	• Ambiguous decision-making process. • Favoritism in code acceptance based on origin and not quality of code and result of peer review.

	• Peer review transparency • Transparency of discussions. • Transparency of promotion to committer or maintainer.	• Discussions with direct impact on project (architecture, development) happen in private.
Development tools	Available and open to all.	• Limited access. • Dependencies on proprietary tools prohibiting non-corporate contributors from participating in the development efforts.
Documentation	Availability of documentations covering architecture, APIs, installation guides, developer guides, development processes, participation guides, tutorials, etc.	• No documentation (source code is documentation) • Poor documentation. • Unmaintained documentation.

CALL TO ACTION

This section is focused on question: How can we do better? When we think of this question, three primary players come to mind:

- Open source developers – create new open source projects, contribute to existing projects
- Head of open source in the enterprise – on behalf of their company, they direct their internal engineers to participate and contribute to open source projects; they decide on open sourcing internal code, and they partner with their peers at other companies on starting new open source projects.
- Open source foundations – such foundations host open source projects, create new open source project in support of their members, mentor developers, advise projects on policy issues, etc.

We believe these three key roles are instrumental in shaping the openness on any open source project. In the following table, we identify some of the actions these players can exert in the various areas that would help an open source project get to a higher level of openness.

	Open Source Developer	**Head of Open Source**	**Open Source Foundation**
License	• Avoid projects with vanity or unclear licenses. • Choose an OSI-approved license for your own project(s). • Communicate the benefits of using a real open source license to your colleagues. • Understand the license you choose for your project or the license of the project(s) you want to participate in.	• Open source code using OSI-approved licenses only. • Mentor company executive on the adoption hurdles a vanity license poses.	• Educate hosted projects on the right choice of license for their projects. • Support selection of an OSI-approved license. • Provide needed protections and ability for companies to collaborate freely for the projects under their auspices. • Act as an agent for the project, receive funds from sponsoring companies, handle trademarks, enforce compliance, provide infrastructure, necessary), support with developer relationships, industry and technical events, driving awareness, etc.

Governance		When establishing new open source projects with industry partners, aim for a balanced governance that give equal footing to all participants – a governance that welcomes contributors and supports a diverse community.	• Advise hosted projects on best governance models. • Help projects to implement their governance.
Access	Foster the culture of free and equal access for everyone.		
Development	• Follow processes. • Recommend improvements. • Suggest new processes.	• Support new projects in creating a number of processes before they launch. These will change over time but it is a huge benefit to have something in place when projects kick off. • Recommend projects to document their processes.	
Community Structure	• Support the right structure for the size of their project. • Recommend improvements based on their own experience participating in the project.	• Set the project governance and structure with growth and scale in mind. • Adopt practices that worked well in other projects. • Build the ability to change as project evolves.	
Releases	• Aim to follow the release cadence when committing to deliver code for a given release. • Evangelize the importance of rhythmic releases. • Provide documentation for your contributions to support good release documentation.	• Promote a given release cadence. • Promote the need for good release documentation. • Promote the need for stable release. Promote experimentation until the project figures out the right cadence and speed.	
Architecture	Design and implement with scale and growth in mind.	Promote for a flexible and modular architecture.	
Communication tools	• Avoid private discussions. • Avoid participating in a closed communication	• Ensure that all newly launched or hosted projects offer communication tools used by typical open source projects and are platform agnostic. • Tools are available for anyone to use them and have access to all of the project's communication.	

	medium (ML, IRC, etc.). • Be inclusive in your communication.		
Transparency	• The project has criteria to promote developers to key positions. • The project has a process that lead to making decisions. • The project has a process to accept incoming code from known entities. • Open communication channels. • Clear governance model.		
Development tools	• Use and promote the best open source tools available to support the project's development. • Mentor new comers into the project on the use of the development tools adopted by the project.	For any new open source projects your company creates, reply on open source development tools that are accessible to everyone.	Ensure that all hosted project rely on development tools that are free and available to everyone.
Documentation	• Document your code. • Contribute documentation explaining architectural decisions, code structure, specific modules or features you have implemented, etc. • Review documentation contributed by others; provide feedback and ideas to improve on them.	• Prioritize documentation as a parallel track to source code development. • Incentivize developers to provide documentation. • Sponsor interns or technical writers to create documentation for open source projects.	

CLOSING

Open source is not restricted to code availability ☺

References

"A New Way of Measuring Openness: The Open Governance Index" by Liz Laffan.

Linux Foundation Enterprise Guides

https://www.linuxfoundation.org/resources/open-source-guides/

TODO Group

http://todogroup.org/

Open Source Compliance in the Enterprise

https://www.linuxfoundation.org/publications/open-source-compliance-enterprise/

ACKNOWLEDGMENT

ABOUT THE AUTHOR

Ibrahim Haddad is VP of R&D and the Head of the Open Source Group at Samsung Research America. Haddad graduated from Concordia University (Montreal, Canada) with a Ph.D. in Computer Science.

He is the author of "Open Source Compliance in the Enterprise" which provides a practical guide on how to best use open source code in products in a legal and responsible way. His latest e-book, "Open Source Audits in Merger and Acquisition Transactions", offers an overview and a guide to open source audits in merger and acquisition transactions, and provides recommendations to improve open source compliance preparedness.

Twitter: @IbrahimAtLinux

Web: IbrahimAtLinux.com