Chapter 12
The Role of Open Source in IoT

Lionel Florit

12.1 The Open Source Movement

Open source in the computer industry is the publishing of source code or hardware design, with associated licensing that permits the reuse, modification, improvement, and potential commercialization under favorable terms. Example of favorable distribution terms includes the following criteria:

- Free distribution: Any party may sell or give away the open source component as part of a larger system without being obligated to pay a royalty or other fee for such sale.
- Source code/design: The source code or design must be distributed and made publicly available.
- Derived works: Derivation and modification of the original open source component are allowed under the original licensing terms.
- No discrimination: The license must not discriminate against any person, group, or a field of business, academics, or research.
- No packaging restrictions: The open source component is not limited to be used as part of a specific distribution or product and is not precluded from being used with other open source or closed source components.
- Technology neutral: There are no assumptions or conditions favoring a specific technology or interface.

While any system can potentially be released under an open source license by its owner, successful open source projects have associated communities of interest that are integral to their success. Such communities are typically geographically distributed and rely on electronic platforms for collaboration. These platforms ensure process compliance, source code management, issue tracking, and continuous integration and test.

The development lifecycle of an open source activity is quite different from the proprietary development cycle. Building a critical mass with an engaged open

A. Rayes, S. Salam, *Internet of Things From Hype to Reality*,
https://doi.org/10.1007/978-3-319-99516-8_12

source community is a critical factor in successful adoption of a project. The ability of a community to garner interest and passion is an indicator of their engagement and potential for providing the advocacy necessary for successful market adoption.

That takes time. On the other hand, if a company decides to create a product, they will staff the project accordingly, and progress in the early phases of the project will be achieved much faster, but the rate of progress will remain relatively constant over time.

However, with open source, once the community is fully engaged, the rate of progress can rapidly accelerate, and the project can potentially progress at a rate that can far outpace closed source development. This is referred to as the "crowdsourcing" effect. According to Howe [5], crowdsourcing is "the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call." Without a doubt, open source is one of the most successful forms of crowdsourcing in the software development industry. Figure 12.1 shows how the crowdsourcing effect impacts the speed of development.

Like other initiatives, the open source movement has certain disadvantages. For example, the leadership of the project does not have control over the contributors. If a key developer decides to move on to another project, there is very little that the coordinators of the open source organization can do. They can't nominate or recruit another leader unless one comes forward. Another issue is focusing the energy of the contributors in the right direction. If a group of people were to make a contribution that is not in line with the original goal or intent of the project, there are only two options: either the leadership rejects the contribution or they allow it. If they reject the contribution, they will lose the potential contributors. If they allow the contribution, they risk diluting the original impact of their open source project.

There are many open source success stories: Linux, Apache Hadoop and HTTPServer, MySQL, Google Chrome, OpenOffice, Android, and Java to name a few. The days of viewing open source as a fad are long gone. Open source is how modern organizations and increasingly more traditional organizations build software. Large corporations are embracing open source and intend to use it in produc-

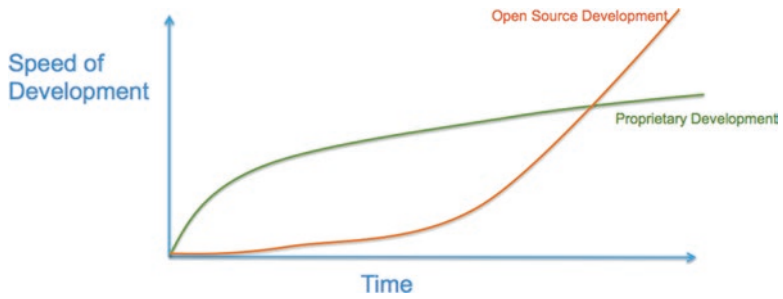

Fig. 12.1 The crowdsourcing effect on the speed of development

tion. Recently, John Donovan, CTO at AT&T, mentioned that, today, open source products represent about 5% of their infrastructure. They plan for that number to reach 50% by 2020. The open source high-speed train is in motion and there is no turning back.

12.2 Why Open Source?

There are numerous reasons driving individuals, corporations, small businesses, nonprofits, government agencies, and other organizations to consume, publish, collaborate on, or support open source. We will discuss the main drivers here.

12.2.1 Drivers for Open Source Consumers

The reasons driving individuals and organizations to leverage and use open source projects are many and can be attributed to the following:

Business efficiency: Many technical problems already have open source solutions available. Hence, instead of wasting time and resources reinventing the wheel, open source consumers can use the best-of-breed solution and focus their efforts on working to address yet-unsolved challenges. These are the types of challenges that add value to their business or mission. This enables a shift from low-value work to high-value work.

Best-of-breed solution: Evidence shows that open source software has better quality compared to closed source [2]. With a closed source system, bugs can be resolved by only the employees of the company developing that system, whereas open source provides clear advantages here: First, it presents the opportunity to tap into a larger pool of contributors and leverage the knowledge of the world's best engineers, not just those on a company's payroll. Second, open source systems are hardened through exposure to a wide array of use cases, not just the one that the original developer intended. This helps in surfacing issues and corner cases much more rapidly compared to traditional test and quality assurance processes baked into typical engineering/development pipelines.

Lower total cost of ownership (TCO): Whether employing open source or closed source systems, certain costs, such as training, maintenance, and support, are sunk costs that have to be paid. In the case of closed source commercial systems, these costs are baked into the equipment price or licensing fees. What sets open source systems apart is the generally lower upfront cost (you don't pay for the right to use the underlying intellectual property). The cost center is shifted from licensing to customization and integration. This generally yields a lower total cost of ownership compared to proprietary and closed systems.

Cost	Open source	Proprietary
Licensing	No	Yes
Training	Yes	Yes
Maintenance	Yes	Yes
Support	Yes	Yes

Modern, nimble development processes: Open source projects go hand in hand with online collaboration tools and platforms that enable distributed, asynchronous, and lock-free electronic workflows. These workflows enable rapid development and allow for more frequent releases. This provides the adopters of open source systems with the required system capabilities without the typical long lead times associated with more traditional corporate processes. This applies not only to new feature functionality but also to bugs and security vulnerabilities. With access to the source code, the adopters of open source systems can often apply patches, or fixes, at their own convenience, without being gated by the release cycles of a specific vendor.

12.2.2 Drivers for Open Source Contributors

Open source contributors include both individuals and large corporations. There are many moral and participatory motivations that drive individuals to contribute to open source projects. While acknowledging the importance of those motives and contributions, in this section, we will only focus on the drivers that encourage large corporations to engage in open source projects.

Workforce multiplier: Open source provides a platform for scaling a development organization's workforce. This happens in two ways: First, when a community comes together to solve a shared challenge, the human capital that becomes dedicated to work on the problem can quickly eclipse what could have been possible in a close corporate setting. Also, the diversity of that capital has been proven to correlate to the degree of innovation and quality of ideas generated. Second, the incubators of the open source system receive peer review and feedback from the community of adopters, who effectively act as "for free" testers of the open source system. This helps improve the original product and bring it to a level of quality and maturity that a small group of developers would have trouble achieving on their own.

Better product architecture: Open source generally leads to well-architected systems that are designed with modularity, maintainability, and flexibility in mind. This is because open source systems, by their nature, are built for a wide array of use cases, environments and users. Hence, technical shortcuts that typically lure developers who are working on proprietary systems, e.g., due to scheduling constraints or laser focus on a specific use case, generally do not manifest in open source projects. Over the long run, this results in greater flexibility and lower customization costs when comparing open source with closed source systems. This is the reason why

some software engineering pundits advocate for architecting all software, even proprietary or internal code, as if it were open source.

Great advertising: Contributors and shepherds of successful open source projects are perceived as industry thought leaders. This bestows upon them the ability to shape the conversation around a particular software problem and allows them to associate their brand with the preferred solution. In a way, this solution becomes the de facto standard for the associated technology. For example, 37Signals is known for creating Ruby on Rails. GitHub is known for creating Hubot.

Customer feedback and trust: Open source offers companies a direct line of interaction with their most passionate customers. It empowers those customers to have a collective powerful voice in the technology development process. The feedback that a company receives can better guide its product development priorities and roadmap decisions, in addition to improving the overall product quality. Furthermore, open source increases transparency which helps promote the customer's trust in a corporation's software.

Attracting and vetting talent: Open source allows a corporation to showcase to the developer community the interesting challenges that it is trying to solve and how it is looking at solving them. Open source developers can casually contribute to projects to learn how the organization works and what it's like to develop solutions for a particular set of challenges. If they are engaged and enthused, the likelihood of them applying for a job at the corporation will be much higher than if the organization were a black box. Similarly, the corporation can see firsthand the quality of the contributed code of prospective employees, which provides better confidence in their capabilities than a typical interview process.

12.3 Open Source vs. Standards

Promoting interoperability through standards is achieved in a very different way compared to open source. Standards organizations come in a continuum of sizes, from the large and well-established international bodies such as IEEE or ITU to the more nimble and usually scope-focused organizations. Smaller organizations tend to have less procedures and target specific problem domains. Regardless of the size of the organization, companies approach them in the same way: they bring their technology and try to turn it into a standard. This usually results in long debates, power struggles, and eventually negotiations, which lead to the creation of a document. This process may take years to conclude. If the company fails to include its technology into a specification, it may try somewhere else, in a different organization.

In the case of IoT, the situation is more complex. The behavior described above is possible, but, since IoT is a green field, some companies may claim that the existing standard bodies do not have the specific skill set or expertise required to realize a new IoT standard. This may result in the creation of a new organization, specifically designed to address one of the IoT verticals such as industrial automation.

However, even if the scene has changed, the format remains more or less the same. A credible standards organization needs to have rules and processes in place to ensure quality and openness. This also applies to IoT standards organizations (Chap. 10). Therefore, the development cycle of IoT standards is on track to match the pace of other technologies in "legacy" standards bodies, and this is to be expected. There needs to be a requirements definition phase, a scoping phase, a debate phase, a drafting phase, a review phase, and finally a voting or some sort of consensus to sanction the work. Eventually, when the standard draft is stable enough, companies can develop to it, which may add several months of delay before a final stable implementation sees the light of day.

In the open source world, however, things can proceed at a much faster rate. A group of developers write source code; they submit it to an existing project if there is one. The code is peer-reviewed. If it doesn't cause any regressions in the system operation and follows the best practice coding guidelines, the code is integrated. No one can block a contribution on the grounds that their company is doing things differently, or because there is a better way to implement. If there is, then code must be submitted by those making such claims. Eventually, the end users will vote by evaluating the code and its functionality. Some user may feel compelled to fix bugs so their company can use the product, and other users benefit instantly.

Of course, the leap of faith a company may take by giving away the implementation of their technology is a substantial barrier to overcome. But the key to success in open source is to add a "secret sauce" that complements the public domain functions. The open source project then becomes a vehicle to get immediate feedback on a way to do things, ignite the spark of curiosity, and attract potential developers and partners. With a common basis built, new proprietary improvements can be added on top of the public domain code. This brings all the players to a higher common ground, which is beneficial for everybody, the producer, and the consumer.

12.4 Open Source Partnering with Standards

As we saw earlier, the way companies approach open source and standards is very different. However, since open source is beneficial for companies, standard bodies quickly realized that they could use open source efforts for their benefit. After all, what the consumer needs is not a 300-page document describing in mundane details how a system should be implemented. Consumers want to have real products in their hand, with real functionalities to use and evaluate in their own business or home environments. This is not something that they get out of the usually dry reading of a standards document. "Code is King," and having some code, which implements a standard, is a very powerful combination. The standard represents an agreement between several parties and the code is the proof that the system on paper does indeed work.

Table 12.1 Examples of open source initiatives for IoT

Standards organization or project	Open source implementation
Open Interconnect Consortium	IoTiVity (Linux Foundation)
oneM2M	IoTDM (Linux Foundation), OCEAN, OM2M (Eclipse)
AllSeen Alliance	Alljoyn
ZigBee® Alliance (IEEE)	Zboss, Open-ZB, NS2, OpNet
CoAP (IETF)	Californium (Eclipse)
MQTT (OASIS)	Mosquitto.org, Paho (Eclipse)
ZWave (Z-Wave Alliance)	openZwave
DASH7 (Alliance)	OSS-7, OpenTag
Modbus (Schneider)	libmodbus.org
BACnet (ASHRAE)	Wacnet
KNX (ISO)	Linknx and Webknx2

Therefore, it is now becoming a must-have for a project under development in a standards body to be associated with some form of open source effort. Following are some examples related to IoT (Table 12.1):

12.5 A Tour of Open Source Activities in IoT

As mentioned previously, the IoT open source community is quite active. There are several open projects; some are backed by consortiums of large industry players; others are backed by just a single startup. Large or small, they all aim at facilitating the deployment of IoT solutions. But, unfortunately, they are not compatible with each other. Some of the larger efforts are attempting to bridge the gap and connect with other overlapping communities or projects.

The list below is far from being exhaustive. It is merely meant to provide an overview of active projects, which have the potential to make a difference in the IoT space. The list is organized per the IoT reference model presented in Chap. 1, Fig. 1.5.

12.5.1 IoT Devices

12.5.1.1 Hardware

12.5.1.1.1 Arduino

Arduino is both a hardware specification for interactive electronics and a set of software that includes an Integrated Development Environment (IDE) and the Arduino programming language. Arduino is “a tool for making computers that can

sense and control more of the physical world than your desktop computer." The organization behind it offers a variety of electronic boards, starter kits, robots, and related products for sale, and many other groups have used Arduino to build IoT-related hardware and software products of their own.

12.5.1.1.2 GizmoSphere

GizmoSphere is an open source development platform for the embedded design community; the site includes code downloads and hardware schematics along with free user guides, specification sheets, and other documentation.

12.5.1.1.3 Tinkerforge

Tinkerforge is a system of open source stackable microcontroller building blocks. It allows the control of motors and reading out sensors with the following programming languages: C, C++, C#, Object Pascal, Java, PHP, Python, and Ruby over a USB or Wi-Fi connection on Windows, Linux, and Mac OS X. All of the hardware is licensed under CERN OHL (CERN Open Hardware License).

12.5.1.1.4 BeagleBoard

BeagleBoard offers credit card-sized computers that can run Android and Linux. Because they have very low power requirements, they're a good option for IoT devices. Both the hardware designs and the software they run are open source, and BeagleBoard hardware (often sold under the name BeagleBone) is available through a wide variety of distributors.

12.5.1.2 Operating Systems

12.5.1.2.1 Contiki

Contiki is an open source operating system for networked, memory-constrained systems with a particular focus on low-power wireless Internet of Things devices. Examples of where Contiki is used include street lighting systems, sound monitoring for smart cities, radiation monitoring systems, and alarm systems. Other key features include highly efficient memory allocation, full IP networking, very low power consumption, dynamic module loading, and more. Supported hardware platforms include Redwire Econotags, Zolertia z1 motes, STMicroelectronics development kits, and Texas Instruments chips and boards. Paid commercial support is available.

12.5.1.2.2 Raspbian

While the Raspberry Pi is not an open source project, many components of its OS are. Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware.

12.5.1.2.3 RIOT

This 1.5kB embedded OS bills itself as "the friendly operating system for the Internet of Things." It fits in the category of Contiki and TinyOS. Forked from the FeuerWhere project, RIOT debuted in 2013. It aims to be both developer- and resource-friendly. It supports multiple architectures, including MSP430, ARM7, Cortex-M0, Cortex-M3, Cortex-M4, and standard x86 PCs.

12.5.2 IoT Services Platform

12.5.2.1 Eclipse IoT Project

Eclipse is sponsoring several different projects surrounding IoT. They include application frameworks and services; open source implementations of IoT protocols, including MQTT CoAP, OMA-DM and OMA LWM2M; and tools for working with Lua, which Eclipse is promoting as an ideal IoT programming language. Eclipse-related projects include:

- Paho provides client implementations of the MQTT protocol.
- Mihini is an embedded Lua runtime providing hardware abstraction and other services.
- Koneki provides tools for embedded Lua developers.
- Eclipse SCADA is a complete Java/OSGi-based SCADA system which provides communication, monitoring, GUI and other capabilities.
- Kura is a Java-/OSGi-based M2M container for gateways. It has support for Modbus, CANbus, MQTT, and other protocols.
- Mosquitto is a lightweight server implementation of the MQTT and MQTT-SN protocols written in C.
- Ponte bridges IoT protocols (MQTT and CoAP) to the web.
- Smarthome provides a complete set of services for home automation gateways.
- OM2M implements the ETSI M2M standard.
- Californium is a Java implementation of the CoAP protocol, which includes DTLS for security.
- Wakaama is an implementation of LWM2M written in C.
- Krikkit is a rules system for programming edge devices.
- Concierge is a lightweight implementation of OSGi Core R5.

12.5.2.1.1 Kinoma

The Kinoma group's hardware and software prototyping solutions help developers, programmers, and designers rapidly create connected products. Owned by Marvell, the Kinoma software platform encompasses three different open source projects. Kimona Create is a DIY construction kit for prototyping electronic devices. Kimona Studio is the development environment that works with Create and the Kinoma Platform Runtime. Kimona Connect is a free iOS and Android app that links smartphones and tables with IoT devices.

12.5.2.1.2 OneM2M the Linux Foundation and Eclipse

The purpose and goal of oneM2M is to develop technical specifications, which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software. oneM2M positions itself as a cross vertical platform. This means that it will be well suited for various sectors such as industrial, energy, home etc. These specifications are being implemented as open source projects at the Linux Foundation (IoTDM), Eclipse (oM2M) and OCEAN.

12.5.2.1.3 Open Interconnect Consortium (OIC)

The goal of OIC is to enable application developers and device manufacturers to deliver interoperable products across Android, iOS, Windows, Linux, Tizen, and more. The Linux Foundation hosts a project called IoTvity, which provides open source code for OIC. At the time of this writing, OIC and oneM2M are specifying gateway functions to bridge the 2 domains.

12.5.2.1.4 IT6.eu, OpenIoT, and IoTSyS

The European Union is actively financing the development of IoT research. OpenIoT and IoTSyS are examples. The OpenIoT website explains that the project is "an open source middleware for getting information from sensor clouds, without worrying about what exact sensors are used." It aims to enable cloud-based "sensing as a service."

IoTSyS is an IoT middleware providing a communication stack for smart devices. It supports multiple standards and protocols, including IPv6, oBIX, 6LoWPAN, Constrained Application Protocol, and Efficient XML Interchange.

12.5.2.1.5 DeviceHive

This project offers a data collection facility for connecting IoT devices. It includes easy-to-use web-based management software for creating devices, applying security rules, and monitoring devices. The website offers sample projects built with DeviceHub, and it also has a "playground" section that allows users to use DeviceHub online to see how it works.

12.5.2.1.6 IoT Toolkit

The group behind this project is working on a variety of tools for integrating multiple IoT-related sensor networks and protocols. IoT Toolkit implements HTTP/REST, CoAP, and MQTT protocols and acts as a stateful bridge between these different protocols.

The primary project is a Smart Object API, but the group is also working on an HTTP-to-CoAP Semantic mapping, an application framework with embedded software agents and more.

Note there is a difference between open source efforts implementing a standard (such as oneM2M and OIC) versus open source efforts trying to realize a middleware implementation with their own data models and protocols. We expect the industry to be more likely to embrace the former.

12.6 Conclusions

There are many aspects to IoT (device, transport, data aggregation and collection, big data, etc.); this translates to a large number of standards and slow progress. Most of the standards are backed by an open source activity. It is now becoming clear that the industry wants to see working code in addition to seeing concise documents describing a technology. The open source community has preceded the standards in most cases, proposing working solutions to real problems.

Therefore there are two classes of open source activities in IoT: one backed by a standard and those evolving by themselves. The latter group is of course more agile and can offer solutions without the overhead of standard development procedures. However, in many cases, there is no domination of one group over the others. This leads to the conclusion that, eventually, a combination of standard plus associated open source will be the long-term solutions the industry will adopt.

Problems and Exercises

1. What is open source? What are the key benefits to the producer and users?
2. Why is open source appealing to software developers? Why is it appealing to application consumers (companies and individuals)?
3. List three downsides for open source projects.
4. List two main disadvantages of open source projects.
5. Linux is a well-known open source project. List three other examples of successful open source projects.
6. Name three examples of IoT open source activities.
7. Are there major differences between standards and open source developments? If so, what is the key difference (i.e., what's the deliverables/outcomes of standard bodies, and are they for open source)? What is the relationship between the two?
8. Name three standards which are implemented in open source.
9. When was the open source label developed? Who developed it?
10. A license defines the rights and obligations that a licensor grants to a licensee. Is it a need for an open source project to provide licenses to its users? What does such license impose?
11. Certification often helps to build higher user confidence. Are there certifications issued for open source? If so, name two examples.
12. "Global Desktop Project" is an example of Open Source initiative developed by the United Nation University. What does it do?
13. What are the main phases of IoT standard development cycles? What are the main phases of IoT open source developments? What are they key differences?
14. It is said that a key to success in open source is to add a "secret sauce" that complements the public domain functions? Why is it the case? Can you provide an example?
15. What is meant by "Code is King" in open source?

References

1. Open Source Definition, The Open Source Initiative. https://opensource.org/osd
2. D. Wheeler, Why Open Source Software (July 2015), http://www.dwheeler.com/oss_fs_why.html
3. http://ben.balter.com/2012/06/26/why-you-should-always-write-software-as-open-source/
4. T. Preston-Werner, Open Source (Almost) Everything (November 2011), http://tom.preston-werner.com/2011/11/22/open-source-everything.html
5. M. Jaksic, et al., *Proceedings of the XIII International Symposium SymOrg,* (2012, June)
6. S. Wright, et al., Open Source and Standards: The Role of Open Source in the Dialogue between Research and Standardization, 2014 IEEE Globecom Workshops (GC Wkshps), (December 2014)
7. Github Hubot. http://hubot.github.com

8. M.St. Laurent, Understanding Open Source and Free Software Licensing. O'Reilly Media. p. 4. (May 2008), ISBN 9780596553951
9. W.T. Verts, Open source software. World Book Online Reference Center. Archived from the original on January 1, 2011
10. R. Rothwell, Creating Wealth with Free Software. (Free Software Magazine, September 2008)
11. E.S. Raymond, Goodbye free software, hello open source, August 2—8, Online.: http://www.catb.org/esr/open-source.html
12. Arduino: Online: http://www.arduino.cc/
13. GizmoSphere: Online: https://en.wikipedia.org/w/index.php?title=GizmoSphere&action=edit&redlink=1
14. Tinkerforge: https://en.wikipedia.org/wiki/Tinkerforge
15. BeagleBoard: http://beagleboard.org/
16. AllJoyn: https://allseenalliance.org/developer-resources/alljoyn-open-source-project
17. Contiki: http://www.contiki-os.org/
18. Raspbian: http://raspbian.org/
19. RIOT: http://riot-os.org/
20. Eclipse IoT Project: http://iot.eclipse.org/
21. Kinoma: http://www.marvell.com/kinoma/
22. oneM2M: http://www.onem2m.org
23. IoTDM: Open-source Project implementing oneM2M specification, https://wiki.opendaylight.org/view/IoTDM:Main
24. OIC (Open Interconnect Consortium): http://openinterconnect.org
25. OpenIoT: http://openiot.eu/
26. IoT Toolkit: https://github.com/connectIOT/iottoolkit/wiki/IoT-Toolkit-Overview-and-links
27. T. Pham, Matthew B. Weinstein, Jamie L. Ryerson, Verint Systems Inc. Easy as ABC: Categorizing Open Source Licenses (June 2010), www.IPO.org