An ANT Analysis of Open-Source Development

A Research Paper submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Philip Renkert
Spring, 2020

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

An ANT Analysis of Open-Source Development

The "American Dream" depicts an entrepreneur or inventor, working alone or with a small team to generate new technologies and profits of their own volition. Technologies fundamental to our modern economy and way of life, however, are transitioning from closed product development, with in-house research and development and private intellectual property, to an open model. Referred to as Open Source Development (OSD), this model "promotes the free access and distribution of an end product" (Technopedia, n.d.), allowing anyone to use, modify, and distribute the design or code in accordance with the product's license. Modern OSD ideology emerged in niche academic communities around the inception of the internet and has since come to dominate many technologies that society takes for granted. As of November 2019, an ongoing study by W3Techs Web Technology Surveys reported that 70.8% of websites are powered by Unix, an open-source operating system (W3Techs, 2019). Android, a mobile operating system based on the open-source Linux kernel, claims over 75% of the worldwide mobile operating system market (GlobalStats, 2019). Arduino microcontrollers and Raspberry Pi computers, both open-source projects, are ubiquitous in projects across educational, scientific, and hobbyist communities (Pearce, 2013). This exploration of OSD employs Actor Network Theory to understand its underlying mechanisms, particularly the actors and interactions therewith which make for a successful open-source project.

Methodology: Actor Network Theory and Open Source Development

This research paper addresses the following research question: "How do actors in the development stage of an open-source project contribute to the project's success?" To answer this question, a network analysis of an OSD case study is performed in the following manner:

The network analysis first employs a lifecycle model of OSD developed by Donald Wynn shown in Figure 1.

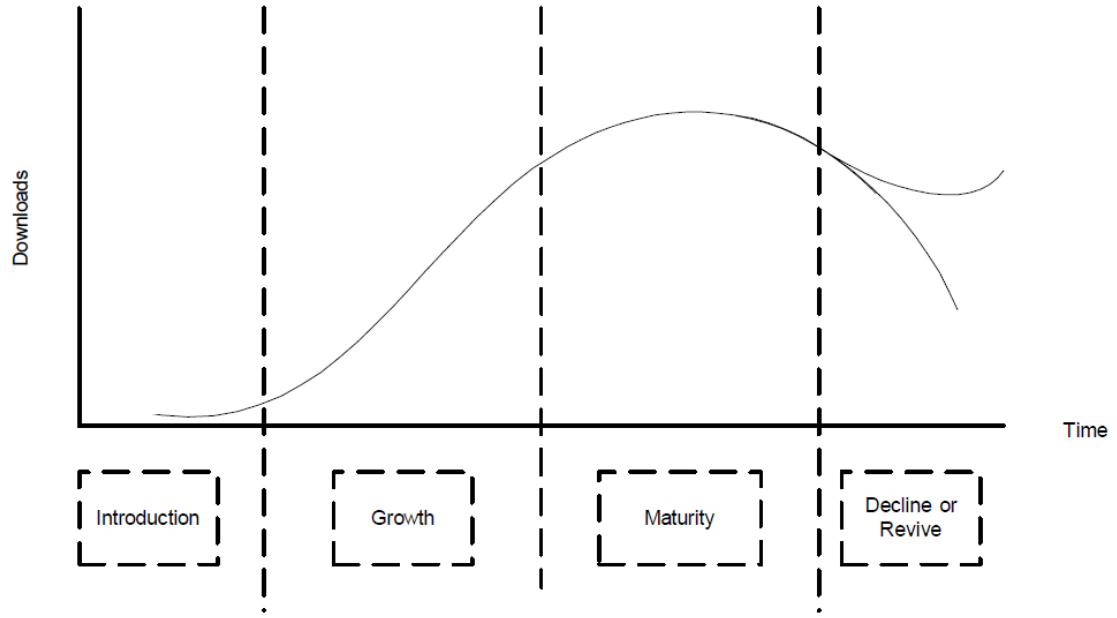

Figure 1. Open-Source Project Life Cycle (Wynn, 2004).

The model discretizes the lifecycle of open-source projects into four stages: 1) introduction, 2) growth, 3) maturity, 4) decline or revival (Wynn, 2004). Analyzing the network at each life stage exposes the network's dynamics. Ideally, one would delineate each stage by measuring community size and vibrancy. To analyze Open Source Software (OSS) projects, Wynn used total downloads as a proxy for community activity.

Critical tasks, events, and activities that shape the project are identified and traced back to the responsible actants and/or interactions. These actants and interactions are then analyzed to uncover motivations, skills, or attributes that combine with other actants to enable the project's success. Here, a successful project is defined as one which achieved the maturity or revival stages of the life cycle and maintains an active community.

Though many projects could have been chosen for the case analysis, the Arduino project dovetails nicely with the purpose of this research. Arduino is an inexpensive and easily-

programmable microcontroller board that makes DIY electronics available to the masses (Kushner, 2011). The project has undergone the primary stages of Wynn's lifecycle model, starting as a simple educational tool in 2005 and maturing to become a stable company (Kushner, 2011). It has a vibrant and growing community that includes not only hackers and hobbyists but also educators, researchers, industry, and derivative products. Arduino is a private company, so much of the data pertaining to Arduino's community is not publicly available. In the absence of first-party data, the term "Arduino" was queried in Google Trends to measure community activity over time. The Google Trends data shown in the following figures measures "Interest Over Time," representing "search interest relative to the highest point on the chart for the given region and time" (Google Trends, 2020a).

Documentary research used to analyze the Arduino network was collected using combinations of the search term "Arduino," "history of Arduino," "open source development," and "open source hardware." Results used in the analysis include a documentary film (Calvo & Alaejos, 2010); company histories and biographies (Barragan, 2016; Kushner, 2011; Torrone, 2011); third-party interviews (Medea, 2013; Stacey, 2017); and first-person accounts from the project's founders (Banzi, 2012; Cuartielles, 2014). Technical references were cited from various sources to further develop how interactions in Arduino's development actor-network manifested in product features and technical details.

The Emergence of Open Source Hardware

Open Source development can be traced back to the mid-20th century, when the advent of computers prompted universities to develop and share software (Longsight, n.d.). The canonical example of open source arrived later in 1991, when Linus Torvalds began his work on the Linux operating system (Welsh et al., 1999). Torvalds combined elements of UNIX and GNU, closed-

source and free operating systems, respectively, and released his work to the internet community. A subsequent snowball of development led to the modern Linux and paved the way for other projects to adopt a similar open model. With the proliferation of OSD projects came organizations who represent open source communities and delineate open-source licenses. The Open Source Initiative provides ten defining criteria for software distributed under an open-source license, covering topics including redistribution, derived works, discrimination, and licensing details (Open Source Initiative, n.d.).

Open Source Hardware (OSH), composed of projects like Arduino, Raspberry Pi, and RepRap 3D Printers, is a younger and lesser-known cousin of OSS. The principles of OSH are derived from those of OSS: "Open source hardware is hardware whose design is made publicly available so that anyone can study, modify, distribute, make, and sell the design or hardware based on that design" (Open Source Hardware Association, 2012). OSH naturally emerged from early making and hacking communities with "long traditions of knowledge-sharing practices" (Gibb, 2014). Recent advances in digital manufacturing technologies have made OSH projects far more viable, as they mitigate the difficulties of turning digital designs into physical products. Machines like 3D printers and CNC mills "allow for the creation of unique, one-off objects without significant setup time or tooling costs" (Mellis & Buechley, 2011).

The Arduino project, one of the most successful examples of OSH, was inspired by a pedagogical need for an easy way to integrate electronics and software into prototype designs. In 2005, A group of teachers and students from the Interaction Design Institute Ivrea (IDII) in northern Italy conceived of a device that combined openly-licensed software and electronics into a package that required no engineering expertise to use (Stacey, 2017). The core Arduino team includes Massimo Banzi, an entrepreneur and associate professor at IDII; David Cuartielles, a

software educator from Malmö, Sweden; Tom Igoe, a professor of computing at New York University; Gianluca Martino, a consultant for a local circuit board factory; and David Mellis, a graduate student at the time (Barragan, 2016; Kushner, 2011). The product's CC BY-SA license allows anyone to sell copies and redesigns; the only caveats being a) attribution must be given to Arduino and b) the derivative product must use the same open license (Stacey, 2017). "If you wanted to, you could take our design, send it to a factory, and order 1,000 boards, and you wouldn't have to pay [Arduino] anything" (Cuartielles, 2014). Though Arduino's reference designs are open, the company trademarked the "Arduino" name and accompanying logo. "Basically, what we have is the brand … and brand matters" (Tom Igoe, quoted in Thompson, 2008). Arduino's innovative model and fervent community has propelled it from a five-man operation into a small multinational corporation, employing 35 people dedicated to the creation of educational experiences in the world of electronics (Cuartielles, 2014).

Actor Network Theory – A Framework for Analysis

This attempt to understand the mechanisms of OSD via network analysis belongs to the field of Science, Technology, and Society (STS), which seeks to understand the "relationship between scientific knowledge, technological systems, and society" (Harvard Kennedy School, 2020). In his book *Two Bits: The Cultural Significance of Free Software*, author Christopher M. Kelty notes how the open source movement has "emerged in tandem with the Internet as both a technical and a social form" (Kelty, 2008). OSD's technical and social forms are tightly integrated. Social communities, comprised of individuals and organizations with complementary skills and similar passions, form around OSD projects. This interplay between social and technical elements results in a social and technical product; OSD cannot be understood from a solely social or technological perspective.

Rooted in STS, Actor Network Theory (ANT) provides an excellent framework to analyze the dynamic and complex networks surrounding Arduino's development. ANT's groundwork was penned by scholars Michel Callon, Bruno Latour, and John Law in the late 20th century (Darryl Cressman, 2009). It is often employed by researchers to understand the infrastructure surrounding technological achievements (David L., 2007). One of its early applications is found in Latour's book *Aramis*, in which Latour studies a failed metro system project in Paris called Aramis. Latour assigns Aramis a voice and human agency, tracing its "life story" back through human and nonhuman actants to the sources of the project's breakdown (Dankert, 2011). Several attributes of ANT render it ideal for an analysis of Arduino and open source development. It is applicable to the complex networks surrounding OSD, and it easily accommodates human and nonhuman actors through the principle of generalized symmetry: "An actor in ANT is a semiotic definition – an actant – that is something that acts or to which activity is granted by another…an actant can literally be anything provided it is granted to be the source of action" (Latour, 1996). The framework includes tools such as Translation for the examination of sociotechnical processes and Punctualization to obtain a clearer view of the network by collapsing static subnetworks into "black boxes" (Darryl Cressman, 2009). These tools allow the network analysis to seamlessly transition between different stages in Wynn's lifecycle model. Inscription is another element of ANT. It describes how technical artefacts embody patterns of use (Monteiro, 2002), and captures how designers, engineers, and scientists often rely on non-human actors in an attempt to enroll other actors in their projects (De Paoli & Storni, 2011).

Like all models that attempt to explain complex systems, ANT is incomplete and subject to criticism. One point of contention is ANT's insistence on the agency of nonhumans; some critics maintain that properties such as intentionality or creativity are distinctly human and

cannot be attributed to nonhuman actors. In a paper in the journal *Social Studies and Science*, Edwin Sayes points to several writers with this contenting stance (Sayes, 2014). Collins and Yearly argue that "social scientists are 'not particularly good' at coming to terms with the competencies of nonhumans and, thus, should leave such an analysis to natural scientists and engineers" (Sayes, 2014). Other writers referenced by Sayes include Khong, Riis, Amsterdamska, and Scaffer (Sayes, 2014). Another criticism of ANT, made by Hans Radder in his "Normative Reflexions on Constructivist Approaches to Science and Technology," is that it relies heavily on observations and case studies, leading to situations where researchers simply report what they find without recognition of general elements like norms and values (Radder, 1992). David Bloor, a fervent critic of ANT, suggests that ANT neither gives proper weight to non-social things and processes nor acknowledges their contribution to social arrangements (Bloor, 1999). ANT has also been criticized for its amorality and failure to account for pre-existing structures or power asymmetries (Walsham, 1997).

Applying the Framework to Arduino

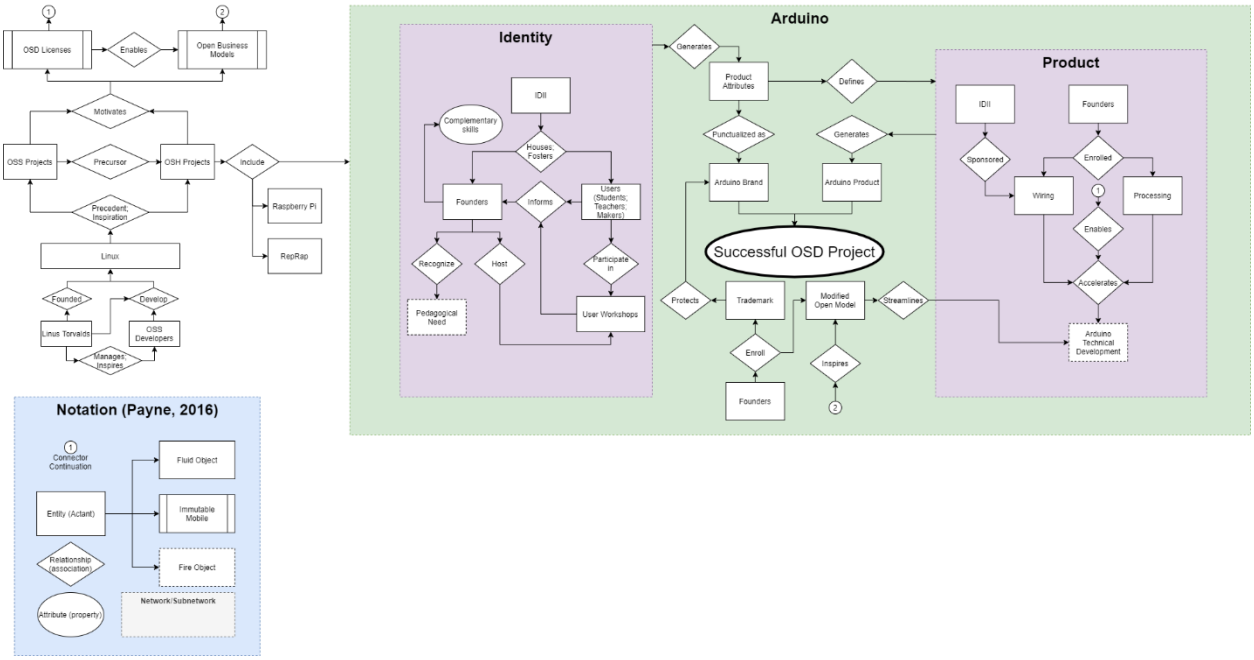

Figure 2. Actor Network Diagram of Arduino's Introduction Phase (Payne, 2016).

Arduino's early development consisted of two subnetworks, the Identity subnetwork and the Product subnetwork, as shown in Figure 2. The former created the vision for Arduino and a set of product features that were punctualized into Arduino's brand. The latter translated Arduino's Identity into physical hardware and built upon previous OSH and OSS projects to accelerate product development. A modified open source model was later inscribed by the founders to protect the project's reputation without sacrificing the benefits of open source. Ultimately, interactions within the two subnetworks resulted in a successful project that stands out among competing products and shows no sign of decline.

Lifecycle Delineation

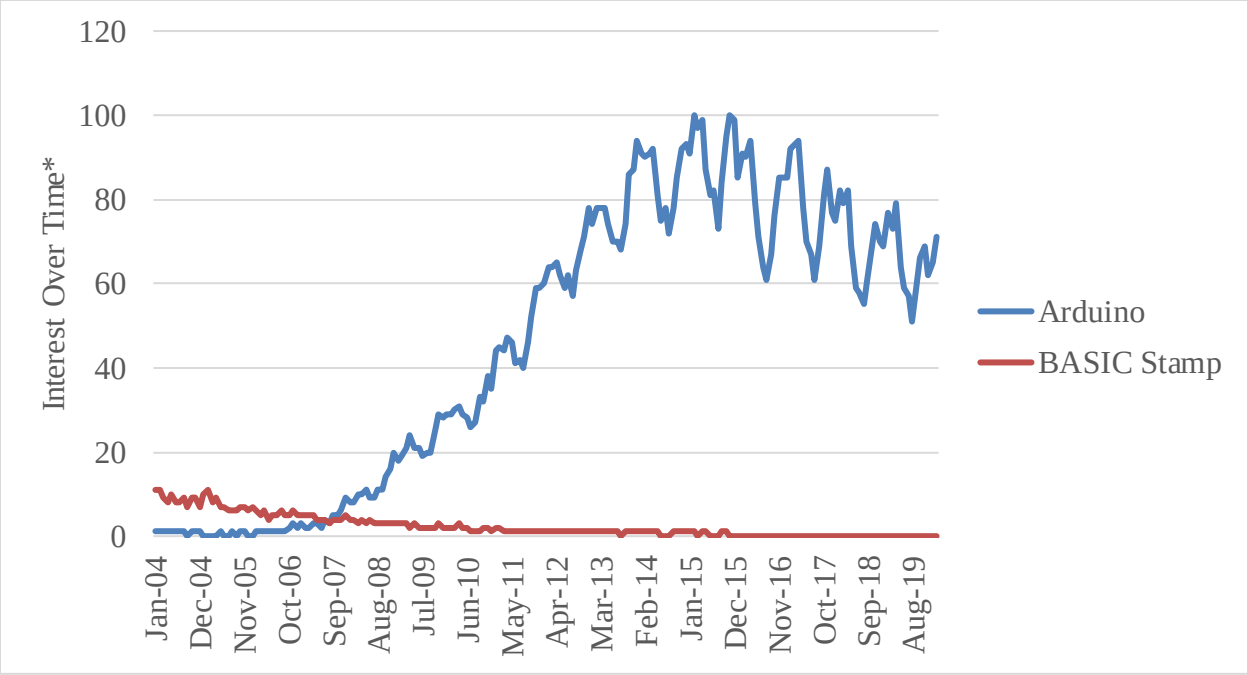

Figure 3. Interest Over Time, Arduino vs. the BASIC Stamp development board *(Google Trends, 2020a)*.

From both quantitative and qualitative perspectives, the Arduino project aligns nicely with the life cycle model outlined by Wynn (Wynn, 2004). The Google Trends data in Figure 3 is used to map the Arduino project onto Wynn's life cycle stages. Data shown in the figure is normalized by the maximum 'Interest Over Time' rating shown (Google Trends, 2020a). For context, Arduino's 'Interest Over Time' is compared with that of the BASIC Stamp development board, a precursor of Arduino. Arduino's slow start, linear growth, and peak in popularity closely match the model curve in Figure 1.

Wynn also provides qualitative attributes of each lifecycle stage derived from his analysis of OSS projects (Wynn, 2004). The Introduction phase occurs from 2005 through 2007, in which Arduino's founders identified a gap between the needs of electronics education and available tools, enlisted team members, and developed initial prototypes. The Growth stage occurs between 2008 and 2013 as users become aware of Arduino and the project acquires a more

formal structure. The project achieved maturity in 2014 when the community size reached a stable equilibrium. At the time of publication, there is no indication of a decline/revival phase in which users find alternative solutions, developers lose interest, and innovation gives way to support and maintenance of existing functionality.

The network analysis begins with Arduino's conception in Winter 2005 and extends to the beginning of the Growth phase in 2007. In this period, two subnetworks, coined the Identity subnetwork and Product subnetwork, develop in parallel to form Arduino and a vision for its future.

Birth of Arduino's Identity

The Identity subnetwork consists of Arduino's early founders, as well as the Interaction Design Institute Ivrea (IDII) and students thereof. Within the sociotechnical context of the early 2000s, actors within this subnetwork transformed an initial need into Arduino's Identity: a vision for the product along with an actionable set of design objectives.

The principle actor in this network is Massimo Banzi, a "bearded and avuncular software architect" who, in 2002, joined IDII as an associate professor to promote new ways of doing interactive design (Kushner, 2011). IDII focuses on designing how humans interact with products, contrasting engineering institutions which emphasize the technical aspects of design. Banzi's courses introduced design and art students to electronics with physical computing projects (Cuartielles, 2014). At the time, the BASIC Stamp development board from California-based company Parallax was the de facto "brain" for such projects (Kushner, 2011). Coded with the proprietary PBASIC language, the Stamp integrated key components including a microprocessor, power supply, memory, and input/output terminals in a tidy package (Parallax Inc, 2020). However, the Stamp was a poor fit for Banzi and his students. Namely, it lacked

computing power for ambitious projects, was expensive at approximately $100 USD per board, and couldn't run on Mac computers (Kushner, 2011). With a limited budget and shrinking class time, Banzi desired a cheap and fast solution.

The Identity network expanded in Winter 2005, when Banzi began discussing the problem with David Cuartilles (Thompson, 2008). The trio of founders was complete when David Mellis, a student at IDII, joined to develop the programming language (Thompson, 2008). These three founders, like the actors in the punctualized Arduino network to come, had complementary skill sets and attributes. In a Wired article by Thompson, Cuartilles describes himself as a "left-leaning academic who's less interested in making money than in inspiring creativity" (Thompson, 2008). Banzi, however, possessed attributes of a canny entrepreneur and foresaw the business potential in OSH (Thompson, 2008). Mellis was a student at the time, providing a unique perspective as both a creator of Arduino and member of its target userbase. Interactions between these three founders and other actors in the subnetwork generated four attributes that are central to Arduino's identity: 1) an unwavering focus on the user, 2) a flexible use case, 3) relentless simplicity and ease-of-use, and 4) a useful bridge between digital and physical realms.

Unwavering User Focus

An uncompromised focus on the user is one of the strongest attributes that emerged from the Identity subnetwork. Arduino's user base punctualized the initial vision for Arduino into the durable eponym in use across the globe today; no product can exist without its users. Each actor in the subnetwork supports this attribute, directly or indirectly. Consider the spirit of the IDII, which promotes user interaction. Speaking of interaction design in his TED talk, Banzi comments "you have to have something that interacts with people" (Banzi, 2012). IDII supplied

the users themselves: design students. Furthermore, these users were students of the design process. Their knowledge of the development process made them ideal candidates to test the product and provide useful feedback to the developers. With direct access to user-centric users in a user-centric institution, the user-centric founders took full advantage of their environment throughout Arduino's development. To test Arduino's ease-of-use and effectiveness, the team "handed 300 blank printed circuit boards to the IDII students with a simple directive: Look up the assembly instructions online, build your own board, and use it for something." (Kushner, 2011). By observing how the students constructed and programmed these early Arduinos, the founders could see exactly what changes were needed to best fit the user. A series of similar workshops took place, with experiments tuned to test different demographics and features. Mellis hosted electronics workshops in his MIT Media Lab to see how easily students and laypeople alike could complete tasks with this beginner-friendly tool. For more user feedback, Arduinos were given away for free to teachers who wanted to explore the possibility of adopting the boards for their classrooms (De Paoli & Storni, 2011). With each experiment and iteration, the design converged to fit the users' needs.

User-centricity manifests in all of Arduino's features, though its multi-platform integrated development environment (IDE) and driver provides a clear example. The software was designed for inclusivity; it works with Mac, Windows, and Linux operating systems. This simple feature aids in enrolling a diverse user community into Arduino's network. Generally speaking, industry and many educational institutions run Windows, programming aficionados run Linux, and designers and artists use a Mac. Had the founders omitted Linux, for instance, they would have neglected a community of advanced users who could contribute to the IDE. Make Magazine contributor Phillip Torrone comments on the subject: "[w]ant freaky cool people

to do neat stuff with your platform? You gotta have your IDE run seamlessly on a Mac and also Linux" (Torrone, 2011).

Flexible Use Case

Users possess the most power in the Arduino network. They not only have the power to influence Arduino's development through workshops, they are given the power to shape Arduino's identity by the way in which they use it. In De Paoli's and Storni's investigation of skill transfer in Arduino's network, the authors speak of how Arduino's developers intentionally inscribed weak forms of control over its users: "The role of the users is only partially defined by the developers … Arduino is intended to be the central core for the implementation and prototyping of interactive products or environments designed by the users themselves" (De Paoli & Storni, 2011). Torrone notes this attribute as well: "[t]he 'what' is still a little vague, and that's Arduino's strength. It's the glue people use to connect tasks together" (Torrone, 2011). Some direct-to-consumer companies sell users a finished pot: they analyze consumers, determine how they will likely use the product, and sell a product designed to be used in that specific way. Arduino sells clay: it is designed to be shaped by the user.

The developers inscribed specific features into Arduino to realize this attribute. For instance, the founders inscribed the avr-gcc compiler, which generates machine-readable instructions for the microcontroller from human readable C++ code (Arduino, 2019). C++ is one of the world's most popular general-purpose programming languages, is well documented, and is easy to learn (W3 Schools, 2020). Its design "give[s] programmers a high level of control over system resources" (W3 Schools, 2020). In other words, the C++ language provides direct access to Arduino's hardware features; it is the perfect language for a board that aspires to be a blank canvas. The Arduino "Library" is another such feature. Libraries are code packages that extend

the Arduino environment by providing extra functionality for working with hardware or manipulating data (Arduino, 2020a). "Standard" libraries are a core set of features built directly into the IDE, but there are also "Contributed" libraries developed by users, accessory manufacturers, and board suppliers (Arduino, 2020a). Anyone with programming skills and a need for new features can leverage Arduino's flexible language and hardware to develop these features themselves. They can then create and share an Arduino library with the rest of the community, punctualizing their code into a black-boxed member of the Arduino network.

Simplicity and Ease-of-Use

From the interactions between founders, students, and teachers in the workshops, simplicity and ease-of-use emerged as a top requirement and core component of Arduino's identity. In their electronics courses, Banzi and Mellis did not want technicalities to obstruct their students' ability to learn. "[W]e worked on Arduino and a lot of other projects [at IDII] to create platforms that would be simple for our students to use, so that our students could just build things that worked, but they don't have five years to become an electronics engineer. We have one month" (Banzi, 2012). Banzi views Arduino's simplicity as a "democratization of engineering," and considers it one of Arduino's most important aspects (Kushner, 2011). This democratization was necessary given Arduino's educational purpose, but it also comes from properties of the founders themselves. None of the founders were electrical engineers who would typically constitute an electronics product development team. Instead, "[t]hey're designers, teachers, artists, and techno-hippies" (Torrone, 2011). They did not appreciate the technology for technology's sake, they saw it as a means to the end of transforming ideas into functional electronics projects. Arduino was a tool they personally wanted: "It was a matter of 'I need this

thing'" (Igoe in Stacey, 2017). In the Arduino network, the identity of the founders, their needs, and users' needs fused and was translated into Arduino's identity.

Again, technologies inscribed into the Arduino product reflect this attribute of its identity. The board is completely "plug and play," for instance, requiring only a USB cable and computer to connect to the pre-configured hardware (Kushner, 2011). Its bare-metal compiler is lightweight, fast, well-tested, and well-understood (Torrone, 2011). When determining what functionality to include with Arduino, the developers opted for a small set of powerful features. "'The challenge is finding a way to accommodate all the different things that people want to do with the platform,' Mellis says, 'without making it too complex for someone getting started'" (Mellis in Kushner, 2011). When users require functionality that cannot be accessed by Arduino's hardware or libraries, they can turn to first or third-party "shields," boards that can be plugged directly into Arduino's printed circuit board (PCB) to expand its capabilities (Arduino, 2020b). Shields help to maintain the critical balance of flexibility and ease-of-use, giving users the option to add complexity and cost to their projects for additional features.

Bridge Between the Digital and the Physical

Arduino's practical purpose is to bridge the divide between digital and physical realms. This attribute emerged from early interactions between founders, students, and teachers, but it also demonstrates how the sociotechnical context surrounding the Identity subnetwork influenced the punctualized product. In the early 2000s, personal computers (PCs) were becoming more specialized and black-boxed, leading to a decreasing interest in 'home-brew' electronics (Kushner, 2011). Additionally, manufacturing technologies had enabled the production of cheap consumer electronics, encouraging the disposal of such devices instead of their repair (Kushner, 2011). A rift had formed between early maker communities and the tactile

world in which they lived; a changing sociotechnical context was confining their medium to the digital realm. Arduino provides a simple interface for software and computers to interact with the real world via sensors and actuators. From an ANT perspective, it translates physical objects into the dynamic and flexible network of software development. Within this network, makers and hackers could again include hardware in their projects, this time at the faster pace and improved flexibility software affords over custom circuits.

Arduino's developers kept this purpose at the forefront of the design. Torrone recognizes a specific example: "[t]he Arduino really took off because it has analog-to-digital input, in other words, you can take in sensor data like light, temperature, sound, or whatever using the low-cost sensors already on the market and get that into the Arduino easily" (Torrone, 2011). Another feature is communication protocols such as serial peripheral interface (SPI) and inter-integrated circuit (I^2C), which allows Arduino to communicate with a plethora of devices including digital sensors and motor drivers (Margolis, 2011). Developers and advanced users of these devices often abstract low-level features with libraries and shields, black-boxing the complexity and making advanced communication capabilities accessible to the broader user community.

Creation of the Arduino Product

As the Identity subnetwork generated Arduino's core attributes and a collection of necessary features, the Product subnetwork encompassed a series of interactions that transformed Arduino's identity into a physical product.

Essentially, the Arduino product is a collection of off-the-shelf components and open source "building blocks," specifically Processing and Wiring. Processing is a designer-friendly, open-source programming language developed in 2001 by Ben Fry and Casey Reas of MIT (Kushner, 2011). Processing allows even inexperienced programmers to create complex and

beautiful visualizations via a simple IDE (Processing Foundation, n.d.), and it was enrolled into the Product subnetwork largely due to familiarity. Faculty and students regularly used the language at IDII, and the Arduino team chose to use it so that students wouldn't have to learn another language in order to program hardware on their prototypes (De Paoli & Storni, 2011). The technology that Processing uses to generate graphics on a screen was flexible enough to code a microcontroller (Kushner, 2011). Because Processing is open-source, Arduino's developers could modify the technology and distribute it in the final product without having to reinvent the wheel.

Wiring is a prototyping platform created by Hernando Barragan, a student of Banzi's in the IDII program (Barragan, 2016). It was Arduino's direct predecessor, with features like a user-friendly IDE and ready-to-use circuit board (Kushner, 2011). Wiring is an open-source project that continues to this day. Instead of joining the Wiring project, the Arduino founders decided to "fork" it, a term used to describe the splitting of an open-source project into two distinct projects (Barragan, 2016). They wanted to make the platform even simpler, inexpensive, and easy to use with a less expensive processor, less memory, and fewer input/output (I/O) pins (Wiring, n.d.).

Wiring is a specific case of the larger role that the IDII played in the Product subnetwork. The environment fostered an accumulation of knowledge through a series of projects and experiences that culminated in Arduino. "When we made Arduino, we built upon years of experience in teaching electronics to beginners" (Cuartielles, 2014). Past projects, whether failures or successes, transferred technical and marketing knowledge to the Founders who later used that knowledge to quickly develop a successful product.

The differentiating benefits of Arduino's Product subnetwork were rapid development and low development cost. "A couple months later, the first Arduino board was ready, paid out of the developers' own pockets" (De Paoli & Storni, 2011). Thompson similarly notes, "In two days, Mellis banged out the code; three days more and the board was complete" (Thompson, 2008). This speed was achieved by strategically translating other projects into the network. Had Wiring and Processing been protected by traditional IP, far more development time and product differentiation would have been required.

Selective Open Source

A common thread between the Identity and Product subnetworks is open source. Arduino uses what I refer to as a "selective open source" model, with traditional open source principles modified by the experience, intuition, and foresight of the founders. As with any network that involves open source, Linux plays a central role. Like Arduino, Linux was born of an "itch that needs scratching" (Thompson, 2008). Torvalds did not like available operating systems, and therefore began to create one himself. When Torvalds shared his code, the hacker community was willing to pitch in and improve it for free, resulting in a virtual workforce that was infinitely bigger and smarter than Torvalds himself (Thompson, 2008). Arduino's founders observed the success that Linus Torvalds, the creator of Linux, realized with OSS. Though there was no OSH precedent to Wiring or Arduino, the potential of opening the project was enough to justify the risk.

The sources used in this analysis point to several reasons why Arduino's founders embraced open source for the hardware designs and software source code. The product network has already exposed a pragmatic reason: it allowed them to modify Processing, Wiring, and other open-source tools and build them into the final product. Wiring's Creative Commons license and

the Copyleft clause in Processings' GNU General Public License require that modified and extended versions of the hardware/software be open as well (De Paoli & Storni, 2011). In an open source network, these licenses are irreversible 'immutable mobiles' enrolled early in the development stage to ensure that future users cannot privatize the project (Free Software Foundation, 2018). Once this mechanism is in place, the network proliferates as future projects stand on the shoulders of those that came before. "[OS projects] like a barn raising in which everyone gets to use the barn. Somebody has a problem and creates a tool to solve it. And once the tool is created, hey—why not share it? The hard work has already been done. Might as well let others benefit" (Thompson, 2008).

Another argument for open source is enhancing users' trust. "Open sourcing makes it easier to trust a product" (Igoe in Stacey, 2017). When the source designs and code are completely transparent, users can evaluate it for themselves to ensure truthful functionality and fair pricing. Open source also bolsters a project's resilience. During Arduino's development, the IDII was running out of funds and preparing to shut down (Kushner, 2011). The founders wanted Arduino to persist beyond themselves and the IDII. In this sense, open sourcing severs the bonds between a project and its founders, translating the project into a new actant with an identity based on the larger community. Additionally, the open source model supports the flexible, blank-canvas attribute of Arduino's identity. As Arduino team member and professor of physical computing at New York University notes: "The open-source nature of Arduino empowers users to modify it and create a lot of different variations, adding on top of what the founders build. This 'ended up strengthening the platform far beyond what we had even thought of building'" (Igoe in Stacey, 2017). Thanks to open source licensing, new ideas, designs, features, and modifications are kept within the ecosystem for others to learn from and build upon.

Despite the advantages of open source, the founders were not idealistic about what the model could achieve. Cuartielles devotes a chapter in the book 'Making Futures,' entitled 'How Deep is Your Love? On Open-Source Hardware,' to the topic. In his discussion, he sheds light on the founder's skepticism of pure open source ideologies and speaks of several 'white lies' that detach open source's perception from reality (Cuartielles, 2014). The first is that radical openness is an illusion. During the design process, especially for hardware projects, someone must make decisions in order to advance the project. In a pure participatory design process, conversations could go on endlessly as the community provides new ideas and debates design aspects. Therefore, in open source networks, founders are often translated to become 'Benevolent Dictators for Life' (a title used in open source communities) to make necessary decisions (Meyer, 2014). Even though Arduino is open source, its development is less about participatory design than the founder's carefully curated knowledge and experience. The students, teachers, and workshop attendants in the Identity subnetwork did not directly control features or technical details. That task was delegated to the founders, who, with the users' best intentions in mind, determined which identity attributes and technical features could flow through the Identity and Product subnetworks. The founders made a conscious decision to control how users informed the design, closed the product development process itself, then released the source designs after the Product subnetwork had been punctualized into a final design. In a radically open design process, Arduino would have been open from the conceptualization phase.

Another area where Arduino differs from traditional open source ideologies is its trademark, which protects the "Arduino" lettering and logo. The punctualized end-state of the Identity subnetwork was the Arduino brand. "Basically, what we have is the brand … and brand

matters" (Igoe in Thompson, 2008). This brand embodies all the attributes of Arduino's identity, and it promises a quality product (Banzi, 2013). To fulfill its goal of being easy to use, the board must work reliably. The Arduino team has lived up to its promise with a failure rate below one percent and a guarantee to replace any defective product. This is the product that Arduino's customers want, but with a purely open source model, they would have no ability to discern whether the board they purchased met these standards. Pure open source allows anyone to manufacture and distribute exact replicas of the original product. Such third-party products, termed "counterfeits" by Banzi, do not guarantee a high level of quality and rarely give back to the open source ecosystem (Banzi, 2013). Inscribing a trademark into the network prevents the diffusion of low quality copies (De Paoli & Storni, 2011). By no means does this prevent other manufacturers from building derivative products, compatible products, or even clones of the Arduino hardware. They just cannot use, and therefore risk, any branding elements which identify official products.

Was the conscious closing of the development process and brand a necessary modification to the pure open source model? The Freeduino project provides a natural experiment to answer this question. The project's website states that "Freeduino is a collaborative open-source project to replicate and publish Arduino-compatible hardware files … [w]hile Arduino is a protected trademark, Freeduino comes with a free and unrestricted license to use the Freeduino name, available for any use" (Freeduino, n.d.). Turning again to Google Trends data, shown in Figures 4 and 5, we can infer that deviations from the pure open source model did not cause a significant proportion of the Arduino community to switch to the Freeduino project.

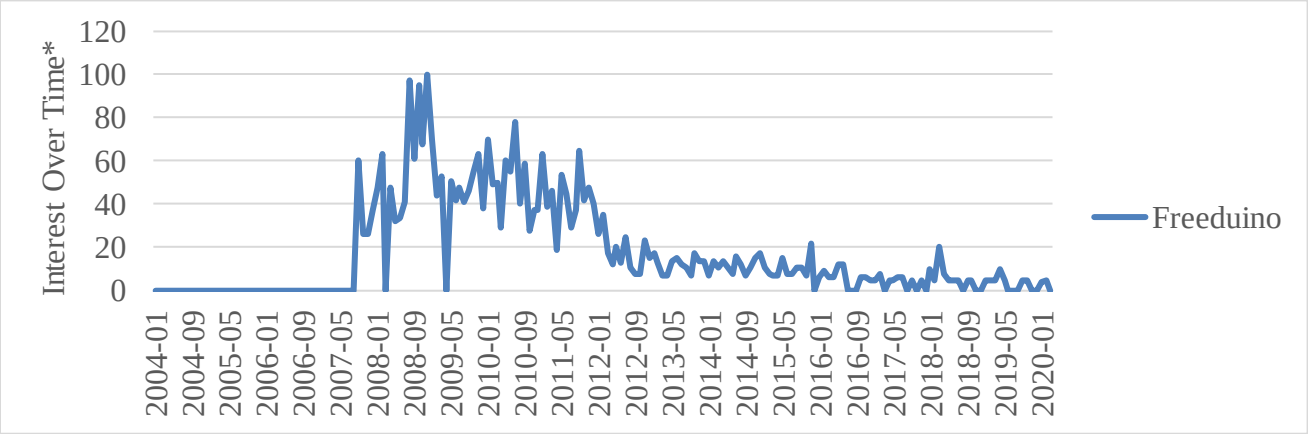

Figure 4. Freeduino, relative interest over time (Google Trends, 2020b).

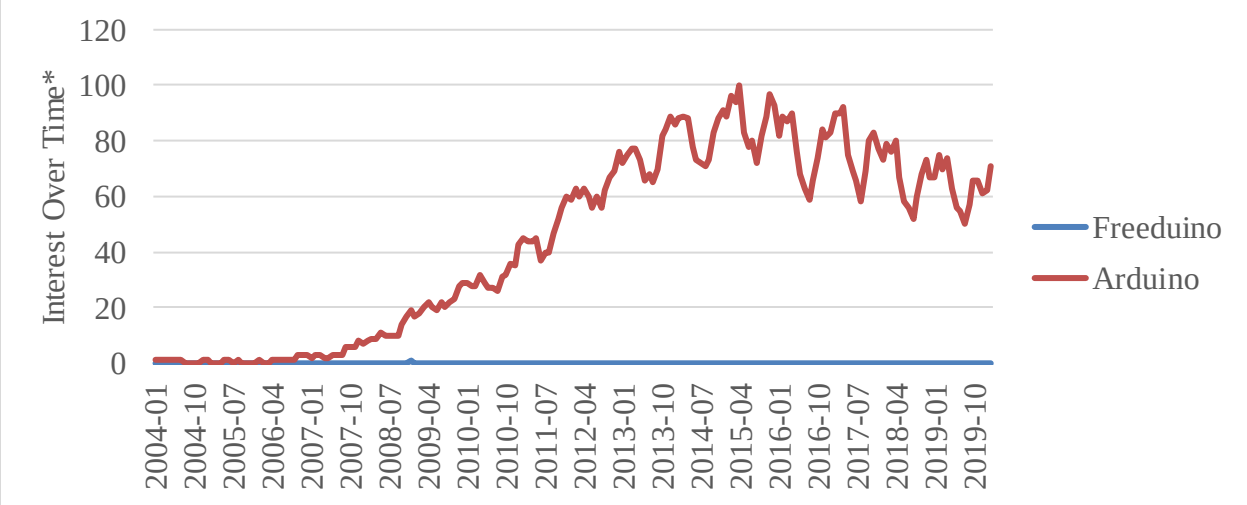

Figure 5. Freeduino interest over time relative to that of Arduino (Google Trends, 2020b).

Limitations

This application of ANT to Arduino's development is limited by a narrow scope and the resources available for analysis. Primarily, conclusions drawn in this study cannot be generalized to the broader case of OSH, a general limitation of ANT. The network analysis focused solely on the Arduino and OSH projects within its immediate network. Furthermore, boundaries had to be drawn around a network that in reality is infinite. For example, the founders of Processing were determined to lie beyond the scope of this paper. Research material used to conduct the network analysis was limited to publicly available documents found using digital applications. No first-party source was available to answer more specific questions about Arduino's network.

Similarly, a very limited amount of data was available to quantify the community size and engagement.

Future Work

Future work on this topic includes extending the analysis into later stages of the life cycle, providing a more complete picture of the network. Additionally, combining results from the network analysis with documentary research of other open-source projects would help to generalize the results beyond the specific case. The STS field could then create models of OSH networks from the generalized results.

Conclusion

The analysis shows that the success of Arduino cannot be attributed solely to the 'magic' of open source, forward-thinking founders, or a strong user community. Complex interactions between the actants in the network were required to translate Arduino from an idea into the ubiquitous tool and vibrant community we see today. Tight integration between founding members, student and teacher users, and institutions like the IDII interacted within the Identity subnetwork, which enrolled a series of workshops to learn from the users and iterate on the product. Ultimately, the Identity subnetwork was punctualized as a brand. The founders intentionally enrolled licenses and trademarks that protected this brand without sacrificing the advantages of open source.

Simultaneously, the Product subnetwork worked efficiently to translate design features determined in the Identity subnetwork into a physical board. Instead of starting from scratch, the developers enrolled Wiring and Processing into the network and modified the technologies to fit their needs. Without open source, this acceleration of the product development process wouldn't have been possible. Actants within the Product subnetwork, Identity subnetwork, and open

source system made independent contributions to the Arduino project, but their coordinated interactions within the larger network created long-term success.

Works Cited

Arduino. (2019). *Build Process*. GitHub. https://github.com/arduino/Arduino

Arduino. (2020a). *Arduino Libraries*. https://www.arduino.cc/en/reference/libraries

Arduino. (2020b). *Arduino Shields*. https://www.arduino.cc/en/Main/arduinoShields

Banzi, M. (2012). *How Arduino is open-sourcing imagination* [TED Talk]. https://www.ted.com/talks/massimo_banzi_how_arduino_is_open_sourcing_imagination

Banzi, M. (2013, July 10). Send in the clones. *Arduino Blog*. https://blog.arduino.cc/2013/07/10/send-in-the-clones/

Barragan, H. (2016). *The Untold History of Arduino* [Blog]. The Untold History of Arduino. https://arduinohistory.github.io/

Bloor, D. (1999). Anti-Latour. *Studies in History and Philosophy of Science Part A*, *30*(1), 81–112. https://doi.org/10.1016/s0039-3681(98)00038-7

Calvo, R., & Alaejos, R. (2010). *Arduino The Documentary* [Documentary]. https://vimeo.com/18539129

Cuartielles, D. (2014). How Deep Is Your Love? On Open-Source Hardware. In *Making Futures*. The MIT Press. https://mitpress.mit.edu/books/making-futures

Dankert, R. (2011, November 30). *Using Actor-Network Theory (ANT) doing research*. Tips over beleid maken, schrijven en uitvoeren. https://ritskedankert.nl/using-actor-network-theory-ant-doing-research/

Darryl Cressman. (2009). *A Brief Overview of Actor-Network Theory: Punctualization, Heterogeneous Engineering & Translation*. School of Communication, Simon Fraser University. http://faculty.georgetown.edu/irvinem/theory/Cressman-ABriefOverviewofANT.pdf

David L. (2007, March 23). Actor-Network Theory (ANT). *Learning Theories*. https://www.learning-theories.com/actor-network-theory-ant.html

De Paoli, S., & Storni, C. (2011). Produsage in hybrid networks: Sociotechnical skills in the case of Arduino. *New Review of Hypermedia and Multimedia*, *17*(1), 31–52. https://doi.org/10.1080/13614568.2011.552641

Free Software Foundation. (2018). *What is Copyleft*. https://www.gnu.org/licenses/copyleft.en.html

Freeduino. (n.d.). *Who Are We?* Https://Freeduino.Org/. Retrieved March 3, 2020, from https://freeduino.org/freeduino_open_designs.html/

Gibb, A. (2014). *Building Open Source Hardware: DIY Manufacturing for Hackers and Makers*. Addison-Wesley Professional. https://proquest.safaribooksonline.com/9780133373912

GlobalStats. (2019, October). *Mobile Operating System Market Share Worldwide*. GlobalStats - Statcounter. https://gs.statcounter.com/os-market-share/mobile/worldwide

Google Trends. (2020a, March 2). *Arduino vs. Basic Stamp: Google Trends*. Google Trends. https://trends.google.com/trends/explore?date=all&geo=US&q=Arduino,%2Fm%2F03m4bv

Google Trends. (2020b, March 7). *Freeduino vs. Arduino: Google Trends*. Google Trends. https://trends.google.com/trends/explore?date=all&geo=US&q=Freeduino,Arduino

Harvard Kennedy School. (2020). *What is STS?* Program on Science, Technology, and Society. http://sts.hks.harvard.edu/about/whatissts.html

Hughes, J. M. (2016). *Arduino: A Technical Reference*. O'Reilly Media, Inc. https://www.oreilly.com/library/view/arduino-a-technical/9781491934319/ch01.html

Kelty, C. M. (2008). *Two Bits: The Cultural Significance of Free Software*.

Kushner, D. (2011, October 26). *The Making Of Arduino*. IEEE Spectrum: Technology, Engineering, and Science News. https://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino

Laporte, L., & Akhtar, I. (n.d.). *Arduino* (No. 110). Retrieved January 30, 2020, from https://twit.tv/shows/triangulation/episodes/110

Laporte, L., & Schwartz, R. (n.d.). *Arduino, the open-source rapid prototyping electronics platform* (No. 61). Retrieved January 30, 2020, from https://twit.tv/shows/floss-weekly/episodes/61

Latour, B. (1996). On actor-network theory: A few clarifications. *Soziale Welt*, *47*(4), 369–381. JSTOR.

Longsight. (n.d.). *History of Open Source*. Retrieved October 15, 2019, from https://www.longsight.com/learning-center/history-open-source

Margolis, M. (2011). *Arduino Cookbook*. http://shop.oreilly.com/product/0636920022244.do

Medea. (2013, April 5). Arduino FAQ - With David Cuartielles. *Medea*. http://medea.mah.se/2013/04/arduino-faq/

Mellis, D. A., & Buechley, L. (2011). Scaffolding Creativity with Open-source Hardware. *Proceedings of the 8th ACM Conference on Creativity and Cognition*, 373–374. https://doi.org/10.1145/2069618.2069702

Meyer, R. (2014, January 17). *On the Reign of "Benevolent Dictators for Life" in Software*. The Atlantic. https://www.theatlantic.com/technology/archive/2014/01/on-the-reign-of-benevolent-dictators-for-life-in-software/283139/

Monteiro, E. (2002, April 1). *Actor-Network Theory and Information Infrastructure*. https://folk.idi.ntnu.no/ericm/ant.FINAL.htm

Open Source Hardware Association. (2012, May 26). *Definition of Open Source Hardware*. Open Source Hardware Association. https://www.oshwa.org/definition/

Open Source Initiative. (n.d.). *The Open Source Definition | Open Source Initiative*. Retrieved October 15, 2019, from https://opensource.org/docs/osd

Parallax Inc. (2020). *BASIC Stamp*. https://www.parallax.com/microcontrollers/basic-stamp

Payne, L. (2016). Visualization in analysis: Developing ANT Analysis Diagrams. *Qualitative Research*. https://doi.org/10.1177/1468794116661229

Pearce, J. M. (2013). Commentary: Open-source hardware for research and education. *Physics Today*, *66*(11), 8–9. https://doi.org/10.1063/PT.3.2160

Processing Foundation. (n.d.). *Processing Overview*. Retrieved March 6, 2020, from https://processing.org/overview/

Radder, H. (1992). Normative Reflexions on Constructivist Approaches to Science and Technology. *Social Studies of Science*, *22*(1), 141–173. https://doi.org/10.1177/030631279202200100 9

Sayes, E. (2014). Actor–Network Theory and methodology: Just what does it mean to say that nonhumans have agency? *Social Studies of Science*, *44*(1), 134–149. https://doi.org/10.1177/0306312713511867

Stacey, P. (2017, September 12). *Made with Creative Commons*. Medium. https://medium.com/made-with-creative-commons/arduino-80b7cad7c006

Technopedia. (n.d.). *What is Open Source?* Techopedia.Com. Retrieved October 28, 2019, from https://www.techopedia.com/definition/3294/open-source

Thompson, C. (2008, October 20). Build It. Share It. Profit. Can Open Source Hardware Work? *Wired*. https://www.wired.com/2008/10/ff-openmanufacturing/

Torrone, P. (2011, February 10). Why the Arduino Won and Why It's Here to Stay. *Make: DIY Projects and Ideas for Makers*. https://makezine.com/2011/02/10/why-the-arduino-won-and-why-its-here-to-stay/

W3 Schools. (2020). *C++ Introduction*. https://www.w3schools.com/cpp/cpp_intro.asp

W3Techs. (2019, November 30). *Usage Statistics and Market Share of Operating Systems for Websites*. W3Techs. https://w3techs.com/technologies/overview/operating_system

Walsham, G. (1997). Actor-Network Theory and IS Research: Current Status and Future Prospects. In A. S. Lee, J. Liebenau, & J. I. DeGross (Eds.), *Information Systems and Qualitative Research* (pp. 466–480). Springer US. https://doi.org/10.1007/978-0-387-35309-8_23

Welsh, M., Kaufman, L., & Dalheimer, K. (1999). A Brief History of Linux. In *Running Linux, Third Edition* (Third). O'Reilly Media, Inc. https://learning.oreilly.com/library/view/running-linux-third/156592469X/ch01s02.html

Wiring. (n.d.). *Wiring Comparison*. Retrieved March 6, 2020, from http://wiring.org.co/hardware/compare.html

Wynn, D. E. (2004). *Organizational Structure of Open Source Projects: A Life Cycle Approach*. 7.