

Roads and Bridges:

The Unseen Labor Behind
Our Digital Infrastructure

WRITTEN BY

Nadia Eghbal

Open up your phone.
Your social media,
your news, your
medical records, your
bank: they are all using
free and public code.




Table of Contents

4	Preface	58	Challenges Facing Digital Infrastructure
5	Foreword	59	Open source's complicated relationship with money
8	Executive Summary	66	Why digital infrastructure support problems are accelerating
11	Introduction	77	The hidden costs of ignoring infrastructure
18	History and Background of Digital Infrastructure	89	Sustaining Digital Infrastructure
19	How software gets built	90	Business models for digital infrastructure
23	How not charging for software transformed society	97	Finding a sponsor or donor for an infrastructure project
29	A brief history of free and public software and the people who made it	106	Why is it so hard to fund these projects?
37	How The Current System Works	109	Institutional efforts to support digital infrastructure
38	What is digital infrastructure, and how does it get built?	124	Opportunities Ahead
46	How are digital infrastructure projects managed and supported?	125	Developing effective support strategies
53	Why do people keep contributing to these projects, when they're not getting paid for it?	127	Priming the landscape
		136	The crossroads we face
		139	Appendix
		140	Glossary
		142	Acknowledgements



Preface



Our modern society—everything from hospitals to stock markets to newspapers to social media—runs on software. But take a closer look, and you'll find that the tools we use to build software are buckling under demand.

Foreword

I stumbled upon the problem described in this report on a hunch. Having previously worked in startups, and then venture capital, I saw the enormous amounts of money being poured into software companies. But as an amateur software developer, I knew that I had never done any of it alone. I used free and publicly available code (also known as “open source” code), which I cobbled together and offered up for personal or commercial purposes. Really, the people behind those projects, whoever they were, had done most of the work.

I mulled over this observation for several years, as I watched the explosion of coding “bootcamps” graduating new software developers left and right, and as I watched startups raise tens of millions of dollars selling products which I knew, under the hood, were probably more public than proprietary code. Having previously worked in the nonprofit sector, I immediately thought of public goods and their associated challenges, yet this vocabulary was strangely absent among my peers in software.

After I left my job in venture capital last year, I set off to explore the paradox I couldn’t stop thinking about: that there were valuable software tools that couldn’t be supported by commercial models, and that they lacked any form of institutional support.

Funnily enough, open source code wasn’t on my original list. I had mistakenly assumed, as had my peers, that these tools were an example of a particularly well-supported public good in software. When I brought up open source to friends and mentors, they gently dissuaded me from pursuing the topic, encouraging me instead to

find other examples that actually needed the help.

A few open source projects crossed my radar, however, and shattered those assumptions. It turned out that sustainability challenges were well-known among those who contributed to open source. The more I dug, the more I found blog posts, articles, and frequent public conversations about the stress and exhaustion felt by those who maintain open source projects. Everybody knew someone else I should talk to, and before I knew it, I had collected countless stories on this topic.

I realized I had walked into a problem with which the producers (open source contributors) were extremely familiar, but that the consumers (software companies and other users of open source code) were seemingly unaware of. That discrepancy made me want to look more closely.

In addition, it seemed that open source itself was changing, perhaps even bifurcating. I found myself having completely different conversations with different generations of open source contributors. They seemed to have divergent philosophies and values; they may as well not have been using the same terminology. I learned that open source had seen an explosion of production as well as demand in the past three to five years, thanks to improvements in developer tools and workflows. Today's open source contributor looked very different from an open source contributor ten years ago, much less thirty years ago. And yet these different generations weren't talking to each other, making productive conversations about sustainability difficult.

A chance conversation with Ethan Zuckerman of the MIT Center for Civic Media gave me an opportunity to share these findings more widely.

I described to Ethan the problem I was seeing, though I didn't know exactly what it all meant or the vocabulary I should be using, and he kindly put me in touch with Jenny Toomey of the Ford Foundation. Jenny suggested I aggregate my findings into a report. In the process, a narrative around our modern digital society, and the hidden infrastructure that powers it, has emerged.

This report would not have happened without Ethan and Jenny taking a chance on a half-baked idea that now, through the process of writing, has been shaped into something more. I am extremely grateful to both of them for their intuition. I am additionally grateful to Michael Brennan and Lori McGlinchey for their guidance, perspective and enthusiasm in the editing process. Finally, and perhaps most importantly, I am indebted to every person working in open source who made their stories public for people like me to read, and especially those who took a moment out of their busy schedules to humor me with a conversation or an email. This report is a collection of their wisdom, not mine. I am particularly grateful for early conversations with Russell Keith-Magee, Eric Holscher, Jan Lehnardt, Andrey Petrov, and Mikeal Rogers, all of whom continue to inspire me with their patience and dedication to open source work. Thank you for your kindness.

Executive Summary

Our modern society—everything from hospitals to stock markets to newspapers to social media—runs on software. But take a closer look, and you’ll find that the tools we use to build software are buckling under demand.

Nearly all software today relies on free, public code (called “open source” code), written and maintained by communities of developers and other talent. Much like roads or bridges, which anyone can walk or drive on, open source code can be used by anyone—from companies to individuals—to build software. This type of code makes up the digital infrastructure of our society today.

Just like physical infrastructure, digital infrastructure needs regular upkeep and maintenance. In the United States, over half of government spending on transportation and water infrastructure goes just to maintenance.¹ But financial support for digital infrastructure is much harder to come by. Currently, any financial support usually comes through sponsorships, direct or indirect, from software companies.

Maintaining open source code used to be more manageable. Following the personal computer revolution of the early 1980s, most commercial software was proprietary, not shared. Software tools were built and used internally by companies, and their products were licensed to customers. Many companies felt that open source code was too nascent and unreliable for commercial use. In their view, software was meant to be charged for, not given away for free.

Today, everybody uses open source code, including Fortune 500

¹
<https://www.cbo.gov/publication/49910>

companies, government, major software companies and startups. Sharing, rather than building proprietary code, turned out to be cheaper, easier, and more efficient. This increased demand puts additional strain on those who maintain this infrastructure, yet because these communities are not highly visible, the rest of the world has been slow to notice. Most of us take opening a software application for granted, the way we take turning on the lights for granted. We don't think about the human capital necessary to make that happen.

In the face of unprecedented demand, the costs of not supporting our digital infrastructure are numerous. On the risk side, there are security breaches and interruptions in service, due to infrastructure maintainers not being able to provide adequate support. On the opportunity side, we need to maintain and improve these software tools in order to support today's startup renaissance, which relies heavily on this infrastructure. Additionally, open source work builds developers' portfolios and helps them get hired, but the talent pool is remarkably less diverse than in tech overall. Expanding the pool of contributors can positively affect who participates in the tech industry at large.

No individual company or organization is incentivized to address the problem alone, because open source code is a public good. In order to support our digital infrastructure, we must find ways to work together. Current examples of efforts to support digital infrastructure include the Linux Foundation's Core Infrastructure Initiative and Mozilla's Open Source Support (MOSS) program, as well as numerous software companies in various capacities.

Sustaining our digital infrastructure is a new topic for many, and the challenges are not well understood. In addition, infrastructure

projects are distributed across many people and organizations, defying common governance models. Many infrastructure projects have no legal entity at all. Any support strategy needs to accept and work with the decentralized, community-centric qualities of open source code. Increasing awareness of the problem, making it easier for institutions to contribute time and money, expanding the pool of open source contributors, and developing best practices and policies across infrastructure projects will all go a long way in building a healthy and sustainable ecosystem.

Introduction

In 1998, a group of security experts in the UK got together to build a free set of encryption tools for the Internet.

Soon everybody was talking about their project, called OpenSSL. (The developers had used an existing Australian project, called SSLeay, as their blueprint.) Not only was it comprehensive and decently reliable, but it was free. Writing cryptography wasn't easy, and OpenSSL had solved a major pain point for developers worldwide.

By 2014, two-thirds of all Web servers were using OpenSSL, enabling websites to securely pass credit card and other sensitive information over the Internet.²

Meanwhile, the project continued to be informally managed by a small handful of volunteers. A security consultant to the U.S. Department of Defense, Steve Marquess, noticed that one contributor, Stephen Henson, was working full time on OpenSSL. Curious, Marquess asked him what he did for income, and was shocked to learn that Henson made one-fifth of Marquess's salary.

Marquess had always considered himself to be a strong programmer, but his skills paled in comparison to Henson's. Like others, Marquess had mistakenly assumed that someone as talented as Henson would have a comfortable salary to match.

Henson had been working on OpenSSL since 1998. Marquess was newer to the project, joining in the early 2000s, and had worked with Henson for several years before learning of his income situation.

2

<http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html>

Having worked with the Department of Defense, Marquess saw how critical OpenSSL was, not just to their software, but to other industries around the world, from enterprise to aeronautics to health care. Until that moment, he had *“always assumed, (as had the rest of the world) that the OpenSSL team was large, active, and well resourced.”*³ In reality, OpenSSL wasn't even able to support one person's work.

3

Email interview with
Steve Marquess

Marquess decided he wanted to help. Although he contributed code occasionally, he realized he could fill a more critical role on the business side. Marquess started out by arranging small consulting contracts through an existing nonprofit to help keep OpenSSL alive in its leanest years.

As the volume of contracts grew, Marquess created the OpenSSL Software Foundation (OSF) to provide an official vehicle for revenue. Despite the number of individuals and companies relying on their software, OSF never received more than \$2,000 in donations per year. Gross revenues (which came from consulting and contract work) never broke \$1M, and much of that went toward security-related testing (which could cost hundreds of thousands of dollars) and server costs.

There was enough to pay the salary of one developer, Stephen Henson. That meant that two-thirds of the Web relied on encryption software maintained by just one full-time employee.

The OpenSSL team continued to work in relative obscurity until April 2014, when a Google engineer named Neel Mehta stumbled upon a major flaw in OpenSSL's software. Two days later, another engineer at the Finnish company Codenomicon discovered the same problem. Both of them immediately contacted the OpenSSL team.

That bug, nicknamed Heartbleed, had been included in a 2011 update. It had gone unnoticed for years. Heartbleed could allow any sophisticated hacker to capture secure information being passed to vulnerable web servers, including passwords, credit card information, and other sensitive data.

Joseph Steinberg, a cybersecurity columnist for *Forbes*, wrote that “some might argue that [Heartbleed] is the worst vulnerability found...since commercial traffic began to flow on the Internet.”⁴

Thanks to wide media reporting, much of the nontechnical world became familiar with the security bug, at least by name. Major services like Instagram, Gmail and Netflix were affected by Heartbleed.⁵ Reporters also drew attention to OpenSSL itself, and how its team had struggled for years to support their work. OpenSSL was a known concern among security experts, but the team did not have adequate resources or attention to address the issues.

Of Heartbleed, Marquess wrote, “*The mystery is not that a few overworked volunteers missed this bug; the mystery is why it hasn’t happened more often.*”

People expressed their support by sending donations to the foundation. Although Marquess was grateful for their enthusiasm, the first round of donations came out to roughly \$9,000: not nearly enough to sustain a team.

4

<http://www.forbes.com/sites/joseph-steinberg/2014/04/10/massive-internet-security-vulnerability-you-are-at-risk-what-you-need-to-do/>

5

<http://mashable.com/2014/04/09/heart-bleed-bug-websites-affected/#01gtseEchaqa>

Marquess took to the Internet to make an impassioned public plea for funding:

“These guys don’t work on OpenSSL for money. They don’t do it for fame (who outside of geek circles ever heard of them or OpenSSL until ‘heartbleed’ [sic] hit the news?). They do it out of pride in craftsmanship and the responsibility for something they believe in.

It takes nerves of steel to work for many years on hundreds of thousands of lines of very complex code, with every line of code you touch visible to the world, knowing that code is used by banks, firewalls, weapons systems, web sites, smart phones, industry, government, everywhere. Knowing that you’ll be ignored and unappreciated until something goes wrong.

***There should be at least a half dozen full time OpenSSL team members, not just one, able to concentrate on the care and feeding of OpenSSL without having to hustle commercial work.** If you’re a corporate or government decision maker in a position to do something about it, give it some thought. Please. I’m getting old and weary and I’d like to retire someday.*⁶

6

<http://veridicalsystems.com/blog/of-money-responsibility-and-pride/>

After Heartbleed, OpenSSL finally got more of the funding it needed—at least for now. They currently have enough money to pay four full-time employees for three years. But a year and a half into that funding, Marquess isn’t sure what will come next.

Marquess said that Heartbleed was a boon for them, admitting it was a “little ironic” that publicity had helped elevate their cause. But after funding runs out and the world moves on, Marquess thinks

they could be back in the same situation as pre-Heartbleed, and perhaps even worse: the client work that took Marquess years to build has dried up, since the team works full-time on OpenSSL right now and no longer has time for contracts.⁷

7

Phone interview with Steve Marquess

Marquess himself is approaching retirement. He is the only person willing to handle the business and operational tasks associated with OpenSSL, including taxes, sourcing client work, and managing donors. The rest of his team prefers to focus on writing and maintaining code. He can't hire someone else into his position when he retires, either, because he currently doesn't take an income. Marquess remarked, "*I don't know that we can hold this together for more than a couple of years*".⁸

8

Email interview with Steve Marquess

OpenSSL's story is not unique, and in many ways, Marquess thinks they are the lucky ones. Countless other projects continue to go unheard of and unsupported. These projects make up the critical digital infrastructure that powers today's software, and in turn, every aspect of our daily lives.

Checking email, reading the news, checking stock prices, shopping online, going to the doctor, calling customer service—whether we realize it or not, everything we do is made possible by projects like OpenSSL. Without them, the technology that modern society relies upon simply could not function.

Many of these projects are built and maintained by volunteers and offered to the public for free. Anyone, from Facebook to an amateur programmer, can use that code to build their own apps. And they do.

If it sounds unbelievable that, as Marquess puts it, a "*ragtag group of*

*amateurs could outcompete huge corporations with their money and resources,”*⁹ consider how this work reflects the rise of peer-to-peer collaboration around the world.

9

Phone interview with Steve Marquess

Unlikely startups like Uber or AirBnB exploded into major corporate powerhouses in just a few years, challenging longstanding industries like transportation and hospitality. Musicians make a name for themselves through YouTube or Soundcloud instead of big record labels. Creative people fund their ideas through crowdfunding platforms like Kickstarter or Patreon.

Similarly, these infrastructure projects sprang from passionate, creative developers who thought *“I could do this better,”* collaborating to build and release code to the world. The difference is that millions of people rely on this code to lead functional daily lives.

Because code is less charismatic than a hit YouTube video or Kickstarter campaign, there is little public awareness of and appreciation for this work. As a result, there is not nearly enough institutional support for the output that sparked an information revolution. But we can’t ignore it for much longer.

In the past five years, our reliance on software, and the free and public code that supports it, has accelerated. Technology has worked its way into every aspect of our lives. And the more people use software, the more software gets built, and the more work is required to maintain it all.


Every successful startup needs public infrastructure to succeed, yet no one company is motivated to act on its own. As the world blazes ahead into a modern age of startups, code and technology,

infrastructure continues to lag behind. The cracks in the foundation are not obvious right now, but they are widening. After years of unprecedented growth that propelled us into a new era of wealth and prosperity, we must act now in order to ensure that the world we built in such a short period of time does not come unexpectedly crashing down.

To understand how to protect our future, first we need to understand software itself.



History and Background of Digital Infrastructure



*"Open source became a movement –
a mentality. Suddenly infrastructure
software was nearly free."*

– Mark Suster, Upfront Ventures

How software gets built

Every website or mobile app we use, no matter how simple, is made up of many smaller components, just as a building is made up of bricks and concrete.

For example, imagine you want to post a photo to Facebook. You open your Facebook mobile app, which triggers Facebook's software to show your news feed.

You upload a photo from your phone, add a comment, then hit "submit." Another part of Facebook's software, responsible for storing data, remembers who you are and posts the photo to your profile.

Finally, a third part of Facebook's software takes the information that you typed into your phone and shows it to all your friends around the world.

Although these interactions take place on Facebook, Facebook did not actually build all the pieces necessary to make it possible for you to post to their app. Instead, they use free, public code, made available on the Internet by volunteers for anybody to use. Facebook does not publicly list the projects they use, but another company they own, Instagram, lists and thanks some of these projects on their homepage and mobile app.¹⁰

Using public code is more efficient for a company like Facebook than building every piece themselves. Building software is like constructing a building. A construction company wouldn't build its hammers and drills from scratch, or source and chop all of the lumber themselves. Instead, it buys the tools from a hardware store, and the

¹⁰

<https://instagram.com/about/legal/libraries/>

lumber from a third-party supplier, to make the job go faster.

Thanks to permissive licenses, companies like Facebook or Instagram are not obligated to pay for this code, but are free to profit handsomely from it. This is not unlike a trucking company (Instagram) using a highway (public code) to transport goods for commercial purposes (Instagram’s app).

Mike Krieger, one of Instagram’s cofounders, emphasized this point in 2013, encouraging other founders to “*borrow instead of building whenever possible. There are hundreds of fantastic [tools]...that can save you time and let you focus on actually building out your product.*”¹¹

11

<https://opbeat.com/blog/posts/picking-tech-for-your-startup/>

Some tools that a software company uses are:

Frameworks: *Software frameworks provide basic scaffolding and structure. Think of it as the blueprint for the entire application. Like a blueprint, a framework lays out how the application might look on mobile, or how information gets saved into the database. Examples include Rails and Django.*

Languages: *Programming languages are the communication backbone of software, like construction workers on a building site using English to communicate. Languages help different software components perform actions and talk to one another. For example, if you create an account on a website and click “sign up,” that application might use the languages JavaScript and Ruby to tell the database to save your information. Popular examples of languages include JavaScript, Python and C.*

Libraries: *Libraries are “prefabricated” pieces of code that make it faster to write software, just as a construction company might buy prefabricated windows instead of building them from scratch. For example, instead*

of a developer writing their own user login system for an application, they can use a library called OAuth. Instead of writing their own code to visualize data on a website, they can use a library called d3.

Databases: Databases store information (for example, user profiles, email addresses, or credit card information) so that it can be used throughout the application. Whenever an application needs to remember something about you, it stores that information in the database. Popular examples of databases include MySQL and PostgreSQL.

Web and application servers: *Web and application servers facilitate various requests that users make on the Internet. They can be thought of as dispatchers or telephone operators. For example, if you type a URL into your browser bar, a Web server will send back the associated page. If you send a message to a friend on Facebook, your message first goes to an application server, which determines who you are trying to contact, then routes your message to your friend's account. Popular examples of Web servers are Apache and Nginx.*

Some of these tools, such as servers and databases, cost money, especially as companies scale. This makes them easier to monetize. For example, Heroku, a cloud-based platform that offers server and database support, offers basic services for free, but charges for higher levels of data or traffic. Heroku powers many major websites, including Toyota and Macy's, and was acquired by Salesforce.com in 2010 for \$212M.¹²

Other types of developer tools, such as frameworks, many libraries, and programming languages, are harder to charge for, and are often built and maintained by volunteers.

Because these types of tools look more like information goods than

12

<http://techcrunch.com/2010/12/08/breaking-salesforce-buys-heroku-for-212-million-in-cash/>

services that can be turned on or off, charging for them would severely limit their adoption. As a result, anyone—whether a billion-dollar company or a teenage coder—can use these components to build their own software for free.

For example, one of the libraries that Instagram uses, according to its homepage, is Appirater. Appirater is a library that makes it easy to remind iPhone users to rate a mobile app. It was created in 2009 by Arash Payan, a freelance developer based in Los Angeles. Payan does not make any income from the project.

It is the equivalent of lumberyards, concrete plants and hardware stores donating their raw materials to a construction company, then continuing to support the company's needs.

How not charging for software transformed society

An expected first reaction is: *“Why did these developers make their software free? Why not just charge for it?”*

The reasons for public software lie in its rich political and social history. But first, let’s examine a hard truth: our society wouldn’t be where it is today if developers hadn’t made it free.

Free software makes it exponentially cheaper and easier to build software.

Uber, a transportation service, recently announced that some developers had built a way to request cars through Slack, a team collaboration app, instead of using Uber’s own mobile app. The project was completed in 48 hours by a team of developers at App Academy, a coding school.

Uber noted that the team was able to get the project done quickly because they *“implemented open libraries such as rails, geocoder, and unicorn [sic] to speed up development and build on a solid foundation.”*¹³

In other words, the amount of coding that the team had to do themselves was greatly reduced because they were able to use free libraries built by others.

Ruby Geocoder, for example, is a library built in 2010 and maintained by a freelance developer named Alex Reisner. Geocoder

13

<https://devblog.uber.com/uber-slack-a-weekend-a-story-of-open-apis/>

makes it easy for an application to look up street addresses and geographic coordinates.¹⁴

14

<http://www.alexreisner.com/about>

Unicorn is a server, built in 2009, which is maintained by a team of seven contributors listed on its website and headed by a developer named Eric Wong.¹⁵

15

<http://unicorn.bogomips.org/CONTRIBUTORS.html>

It's easier than ever to build new software, because there are more prefabricated pieces of code to draw from. To return again to the construction metaphor instead of constructing every piece of a building from scratch, one can simply buy a prefabricated framework, foundation and walls, then put them together like Legos.

As a result, new developers are minted every day, even if they themselves don't necessarily know how to build the tools from scratch. The Bureau of Labor Statistics expects the number of employed software developers to rise 22% from 2012 to 2022—much faster than average, compared to other occupations.¹⁶

16

<http://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm>

Free software is directly responsible for today's current startup renaissance.

The cost of starting a company has dropped dramatically since the first dotcom boom in the late 1990s. Venture capitalist and former entrepreneur Mark Suster reflected on his experience in a 2011 blog post:

“When I built my first company starting in 1999 it cost \$2.5 million in infrastructure just to get started and another \$2.5 million in team costs to code, launch, manage, market & sell our software. [...]”

The first major change in our industry was imperceptible to us as an industry. It was driven by the introduction of open-source software, most notably what was called the LAMP stack. Linux (instead of UNIX), Apache (web server software), MySQL (instead of Oracle) and PHP. Of course there were variants – we preferred PostGres to MySQL and many people used other programming languages than PHP.

Open source became a movement – a mentality. Suddenly infrastructure software was nearly free. We paid 10% of the normal costs for the software and that money was for software support. A 90% disruption in cost spawns innovation – believe me.¹⁷

The availability of free software components today (as well as cheaper hosting and cloud services, like Amazon Web Services and Heroku) means that a technology startup no longer requires millions of dollars to get off the ground. Entrepreneurs can conceivably release a product and find a market without spending a single dollar, then raise money from venture capitalists only after they've shown strong signs of demand.

Alan Schaaf, the founder of Imgur, a popular image-sharing site and one of the top 50 most-trafficked sites in the world, famously said that the only money he ever spent to start the company was seven dollars to purchase the domain name. Imgur was profitable, and Schaaf did not take any outside money for 5 years before raising \$40 million from VC firm Andreessen Horowitz in 2014.¹⁸

Venture capitalists and other institutional investors, in turn, have started writing smaller checks to companies, giving rise to new

17

<http://www.bothsidesofthetable.com/2011/06/28/understanding-changes-in-the-software-venture-capital-industries/>

18

<http://articles.latimes.com/2014/apr/03/business/la-fi-tn-imgur-40-million-funding-20140402>

subsets of investing, including:

Seed stage: *Venture firms providing the first round of funding, rather than later-stage growth capital*

Micro VCs: *Venture firms loosely defined as less than \$50 million under management*

Accelerators: *Firms that provide small amounts of capital, often less than \$50,000, as well as advice and mentorship to early-stage companies*

\$10M can fund a hundred companies today, compared to one or two in the 1990s.

Free software made it easier for people of all demographics to learn to code, making technology accessible to the world.

If you wanted to learn how to code at home today, you might start by learning Ruby on Rails. Rails is a popular software framework and Ruby is a programming language. Anyone with Internet access can install these tools on any computer for free. Because they are free, they are also very popular, which means there is plenty of information online to help you get started, from formal tutorials to question-and-answer forums. This means that learning how to code is as accessible as teaching oneself to read and write English or French.

By comparison, software frameworks and languages that were not open source required paying for access, using specific operating

systems or other tools, and agreeing to licensing constraints that could affect patents for any software built using the framework. Today, it is difficult to find examples of frameworks and languages that are not open source. One of the most famous examples of a proprietary software framework is .NET, developed and released in 2002. In 2014, Microsoft announced that they were releasing a version of .NET as an open source project, called .NET Core.

Audrey Eschright, a software developer, wrote about how open source software helped her learn to code as a teenager in the late 1990s:

“I wanted to learn to program but I didn’t have money. Not the college student version of not having money—my family situation was low-income, but also highly chaotic....This is going to seem strange to anyone [today], but at the time there were basically two options for someone who wanted to write real software: you could use a PC with Windows and pay extra for Microsoft’s development tools, or you could have access to a Unix system and use gcc....So my goal became to get access to accounts on Unix systems so I could learn how to write code and do cool stuff.”¹⁹

19

http://lifeofaudrey.com/essays/love_and_money.html

Jeff Atwood, a longtime .NET developer, described his decision to use Ruby for a new software project, Discourse, in 2013:

“Getting up and running with a Microsoft stack is just plain too hard for a developer in, say, Argentina, or Nepal, or Bulgaria. Open source operating systems, languages, and tool chains are the great equalizer, the basis for the next great generation of

*programmers all over the world who are going to help us change the world.*²⁰

20

<http://blog.codinghorror.com/why-ruby/>

With the explosion of startups have come a number of initiatives to teach people to code, whether they are children, teenagers, underserved minorities, women or career switchers. Some examples include Women Who Code, Django Girls, Black Girls Code, One Month and Dev Bootcamp.

Some of these organizations are free, while others charge tuition. All of them rely upon free software to teach their students. For example, Django Girls has taught over 2,000 women to code, in 49 countries around the world.²¹ Although the organization did not develop Django themselves, they are able to use Django, which students download and use for free, in their curriculum

21

<https://djangogirls.org/>

Dev Bootcamp teaches career switchers to code, preparing everyone from English teachers to military veterans to become professional software developers. The program costs \$12-14,000. Dev Bootcamp teaches Ruby, JavaScript, Ruby on Rails and SQL, among other components. All of these components are free for students to download and use, and Dev Bootcamp does not have to pay to use these materials. Dev Bootcamp was recently acquired by Kaplan for an undisclosed sum in 2014.²²

22

<https://www.edsurge.com/news/2014-06-25-dev-bootcamp-no-longer-bootstrapped-acquired-by-kaplan>

If such critical pieces of software were not free, people from all walks of life would not be able to take part in today's technology renaissance. There are still numerous social and economic barriers that prevent many more from participating, as well as costs associated with physical equipment like laptops and an Internet connection, but the programming tools themselves do not cost money.

A brief history of free and public software and the people who made it

Now that we've covered how making software free benefits society, let's look at how the software itself came about.

Although we've used the term "free software" to refer to software that does not cost any money to its users, the term "free software" is actually a highly contextualized term that refers specifically to the software's license properties. Free software advocates emphasize that "free" refers to a political freedom rather than the price, and sometimes use the Spanish word *libre* (meaning freedom, as opposed to *gratis*, the Spanish word for free price) to clarify the distinction.

In the 1970s, when computers were still a nascent technology, programmers had to build their own computers and write custom software themselves. Software was not yet standardized and was not considered to be a monetizable product.

In 1981, IBM introduced the "IBM PC," or "Personal Computer," bringing hardware to a mass market. Within a couple of years, custom computer setups fell away as everybody adopted the IBM standard. IBM became the dominant computer within a highly fractured personal computer market, capturing over half of market share by 1986.²³

Along with standardized hardware, then, came an opportunity for standardized software. Suddenly everyone wanted to turn software into a business. IBM hired a then-unknown company called Microsoft to write the operating system for its new PC. That operating system,

23

<http://arstechnica.com/business/2012/08/from-altair-to-ipad-35-years-of-personal-computer-market-share/2/>

MS-DOS, was released in 1981. Other companies began to follow suit, offering software under commercial licenses. These licenses prevented the user from copying, modifying or redistributing the software.

Proprietary software still exists today: for example, Adobe Photoshop, Microsoft Windows, or GoToMeeting. While proprietary software can be profitable for the company that builds and licenses the product, its restrictions also limit its scope and distribution. Any changes to the software's design or implementation have to originate from the company itself. And proprietary software is expensive, often costing hundreds of dollars and permitting the designated purchaser to use only that copy.

Understandably, some computer scientists felt concerned about the closed and proprietary direction that software was taking, believing that it undermined the true potential of software. Richard Stallman, a programmer at the MIT Artificial Intelligence Laboratory, felt particularly strongly about the need for software to be free and modifiable.

Over the next couple of years, as several of his colleagues began working on proprietary software projects, Stallman felt he could not ignore the situation any longer. In 1983, he launched GNU, a free operating system, and in doing so sparked what came to be known as the “free software movement,” which galvanized a group of people who believed that software could have a greater reach and benefit to society if it were made freely available. Stallman later founded the Free Software Foundation in 1985 to support GNU and other free software efforts.

The Free Software Foundation defines free software as “software

that gives the user the freedom to share, study and modify it.”²⁴ GNU defines four freedoms associated with such software:

24

<https://www.fsf.org/about/what-is-free-software>

Freedom 0: *The freedom to run the program as you wish, for any purpose.*

Freedom 1: *The freedom to study how the program works, and change it so it does your computing as you wish.*

Freedom 2: *The freedom to redistribute copies so you can help your neighbor.*

Freedom 3: *The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes.*²⁵

25

<https://www.gnu.org/philosophy/free-sw.html>

The free software movement was, and continues to be, deeply rooted in social advocacy. In 1998, when Netscape released the source code for its popular browser, the conversation began to shift from politics to technology.

Some technologists believed that focusing on the practical benefits of free software would help bring its message to a wider audience. For example, they pointed out that free software was cheaper to build and could lead to superior software, because the public can find bugs and contribute fixes. This type of pragmatism was distinct from the moral obligation that Stallman and his supporters believed they had to promote free software.

These technologists gathered in Palo Alto for a strategy session. Christine Peterson, a nanotechnologist in attendance, suggested the term “open source.”²⁶ Shortly after, two attendees, Bruce Perens and Eric Raymond, created the Open Source Initiative.

26

<http://opensource.org/history>

Software whose source code is publicly available is called “open source.” It is analogous to being able to open up the hood of a car and see what’s inside, instead of having the engine sealed off from view. Open source licenses always include a provision that allows the public to use, modify, and redistribute the code. In this sense, there is no legal difference between free software and open source licenses. Indeed, some people have called open source a “marketing campaign” for free software.

However, the most important distinction is the differing cultures that each movement created. The open source software movement broke away from the social and political associations with free software by instead focusing on the practical benefits of software development and encouraging wider creative and business applications. As Stallman himself wrote, “*Open source is a development methodology; free software is a social movement.*”²⁷

27

<http://www.gnu.org/philosophy/open-source-misses-the-point.en.html>

Although “free software” and “open source software” are often discussed together, they are politically distinct, the former being more closely associated with ethics and the latter with pragmatism. (The remainder of this paper will use the term “open source” to emphasize the critical role it plays in software infrastructure.)

Open source created space for growing distinctions and styles of software development, free from ethical complexities. One organization might release its source code to the public, but only accept

changes from a couple of contributors. Another organization might require that the code is developed in public and accept changes from anyone, so that more people could take part in the process. In 1997, Raymond wrote an influential essay called *The Cathedral and the Bazaar* (later published as a book in 1999) which explored these styles.

Today, open source has become a popular software practice for many reasons, in terms of both efficiency and cost. It's also how much of digital infrastructure gets built. We've discussed how making this software more freely available has benefitted all of society, but open source has benefits for its creators, as well.

Open source is cheaper to build.

Before open source software existed, technology firms treated software like any other paid product: a team of employees built new software internally, then sold it to the public. While this meant software had a clear business model, it also came with increased development costs. Proprietary software requires a full-time paid team to support its development, including developers, designers, marketers, and lawyers. It's far cheaper to simply crowdsource software, built and maintained by a community of volunteer developers.

Open source is easier to distribute.

People are more likely to adopt software that is free to use and modify than software that costs hundreds of dollars to license and was developed in a black box. Not only will developers want to use it for free, but they might be inclined to tell their friends to use it as

well, amplifying the effects of its distribution.

Open source is flexible to customize.

Open source software is free to copy and modify for one's own purposes, with various levels of permissiveness. This means that if a developer wants to make improvements to a piece of software, he or she can copy the project and change it. (This practice is called “forking.”)

Many popular projects started as a modification of an existing piece of software, including *WordPress* (content management system that powers 23% of the world's websites²⁸), *PostgreSQL* (one of the world's most popular and fast-growing databases²⁹), *Ubuntu* (operating system used by 10% of the world's websites³⁰), and *Firefox* (one of the most popular web browsers in the world³¹).

WordPress began as an offshoot of an existing blogging project, b2 (also known as cafelog). Two software developers, Matt Mullenweg and Mike Little, decided they wanted a better version of b2 and subsequently forked the project. Mullenweg decided to fork b2, rather than another project called TextPattern, because b2's licenses were more permissive. His original thought process from 2003 is described below:

“What to do? Well, *TextPattern* looks like everything I could ever want, but it doesn't look like it's going to be licensed under something politically I could agree with. Fortunately, *b2/cafelog* is GPL [GNU General Public License, a free software license], which means that I could use the existing codebase to create a fork.[...]

28

<https://en.wikipedia.org/wiki/WordPress>

29

<http://www.zdnet.com/article/as-dbms-wars-continue-postgresql-shows-most-momentum/>

30

<http://w3techs.com/technologies/details/os-ubuntu/all/all>

31

<https://www.netmarketshare.com/browser-market-share.aspx?qprid=0&qpcustomid=0>

*The work would never be lost, as if I fell off [sic] the face of the planet a year from now, whatever code I made would be free to the world, and if someone else wanted to pick it up they could.*³²

32

<http://ma.tt/2003/01/the-blogging-software-dilemma/>

If software were developed in a closed, proprietary environment, developers would have no ability to change that software, unless they worked at the company. If they tried to build their own improved version to imitate the original, they might face intellectual property concerns. With open source software, the developer can simply change the software him- or herself and release it to the public, as Mullenweg did. Open source software, then, enables rapid proliferation of ideas.

Open source gives employees more bargaining power.

Software takes time to learn, whether it's a new programming language or framework. If every company used a proprietary set of tools, developers would be less inclined to change jobs, because their technical skills only apply to that one place of employment. They would have to be retrained in a new technology at their next place of employment.³³

33

Thanks to Karl Fogel for reminding me of this benefit.

When companies use open source technology, a developer has a reusable set of skills, which leads to more freedom to work wherever he or she prefers. For example, multiple companies might use the same Ruby programming language in their software. In addition, if the company's product itself is open source, the output belongs to the developer as much as it does the company. The developer can take their work with them if they choose to leave the company (versus, for example, being constrained by a non-disclosure

agreement, if the code were proprietary). All of these benefits give the employee more agency than he or she would have had with proprietary software. Many companies today advertise their use of open source software as a recruiting tactic, because it favors the developer.


Open source has the potential to be more stable and secure.

Theoretically, when a software project has many active contributors and a thriving community, the code should be less vulnerable to security flaws and disruptions in service. That's because more people would ideally be reviewing the code, looking for bugs and fixing any problems that they see. By contrast, in a proprietary software environment, the only people who would see the code would be the team of people developing it. Instead of, say, 20 employees looking at the code at Oracle, a popular open source project could have 2,000 volunteers reviewing the code for vulnerabilities. (Note that this belief does not always match reality, and has created the opposite problem: people mistakenly believing that more people are reviewing open source software than actually are, when in reality nobody is taking responsibility. This will be discussed in a later section.)

Open source software clearly has a number of benefits. How do these projects collectively fit into a broader ecosystem?



How The Current System Works



"A lot of [our] members work in tech, either on the web or on software. As a result, they work on things that don't last very long."

– Tim Hwang, Bay Area Infrastructure Observatory

What is digital infrastructure, and how does it get built?

Earlier in this report, we compared building software to constructing a building. Those public software components are what collectively form our digital infrastructure. To understand this concept, consider how physical infrastructure works.

Everybody relies upon a number of physical infrastructure projects to facilitate our day-to-day lives. Turning our lights on, driving to work, washing dishes: we may not often think about where our water, roads or electricity come from, but we have physical infrastructure to thank. Private and public partners work together to build and maintain our transportation, sewage, water, electric, and communication systems.

Similarly, although we do not often see or think about the apps and software we use on a daily basis, all of them rely upon free and public code to function. Together, in an increasingly digital society, these open source projects make up our digital infrastructure.

However, there are several major differences between physical and digital infrastructure, which affect how the latter is built and sustained. In particular, there are differences in *cost*, *maintenance*, and *governance*.

Digital infrastructure is faster and cheaper to build.

Building physical infrastructure is notoriously expensive. These

projects are physically large in scale and can take months or years to complete.

The United States federal government spent \$96 billion on infrastructure projects in 2014, and state and local governments spent a combined \$320 billion in the same year. Slightly less than half (43 percent) of that spending went towards new construction; the remainder was spent on operations and upkeep of existing infrastructure.³⁴

34

<https://www.cbo.gov/publication/49910>

Proposing and funding new physical infrastructure projects can be an extended political process. Transportation funding has been a contentious topic in the United States for the past decade, where the federal government faces a \$16 billion shortfall for transportation funding.³⁵ U.S. Congress recently passed the first multi-year transportation bill in a decade, setting aside \$305B for highways, after years of political obstacles that prevented funding infrastructure from being funded more than two years at a time.³⁶

35

<http://thehill.com/policy/transportation/255264-mc-carthy-were-going-to-make-sure-we-get-the-highway-bill-done>

36

<http://www.wsj.com/articles/house-passes-five-year-transportation-bill-1449167609>

Even after a new infrastructure project has been earmarked and funded, it can take years to complete, fraught with uncertainties and unforeseen obstacles. The Central Artery/Tunnel project in Boston, Massachusetts, also known as the Big Dig, took nine years from planning to initial construction. Its projected cost was \$2.8 billion, with a scheduled completion date for 1998. In reality, the project ended up costing \$14.6 billion and was not completed until 2007, making it the most expensive highway project in the United States.³⁷

37

https://en.wikipedia.org/wiki/Big_Dig

By contrast, digital infrastructure does not have any of the costs associated with building physical infrastructure, such as zoning a location or purchasing materials. This makes it easy for anyone to

propose a new idea and get started in very little time.

MySQL, the second most popular database in the world³⁸ and part of a critical collection of tools that helped launch the first tech boom, was published by its authors, Michael Widenius and David Axmark, in May 1995. It took less than two years to develop.³⁹

38

<http://www.zdnet.com/article/as-dbms-wars-continue-postgresql-shows-most-momentum/>

39

<https://en.wikipedia.org/wiki/MySQL>

Ruby, a programming language, took less than three years from its initial conception in February 1993 to public release in December 1995. Its author, computer scientist Yukihiro Matsumoto, decided to create the language after a conversation with his colleagues.⁴⁰

40

[https://en.wikipedia.org/wiki/Ruby_\(programming_language\)](https://en.wikipedia.org/wiki/Ruby_(programming_language))

Digital infrastructure changes frequently.

Because digital infrastructure is so cheap to build, the barriers to entry are lower, and software tools change more frequently.

Physical infrastructure is built to last, which is partially why these projects take so long to plan, fund, and build. The London Underground, London's public rapid transit system, was built in 1863; the underground tunnels dug for the subway system are still in use today.⁴¹ The Brooklyn Bridge, which connects the boroughs of Brooklyn and Manhattan in New York City, was completed in 1883 and did not undergo any major renovations until 2010, over one hundred years later.

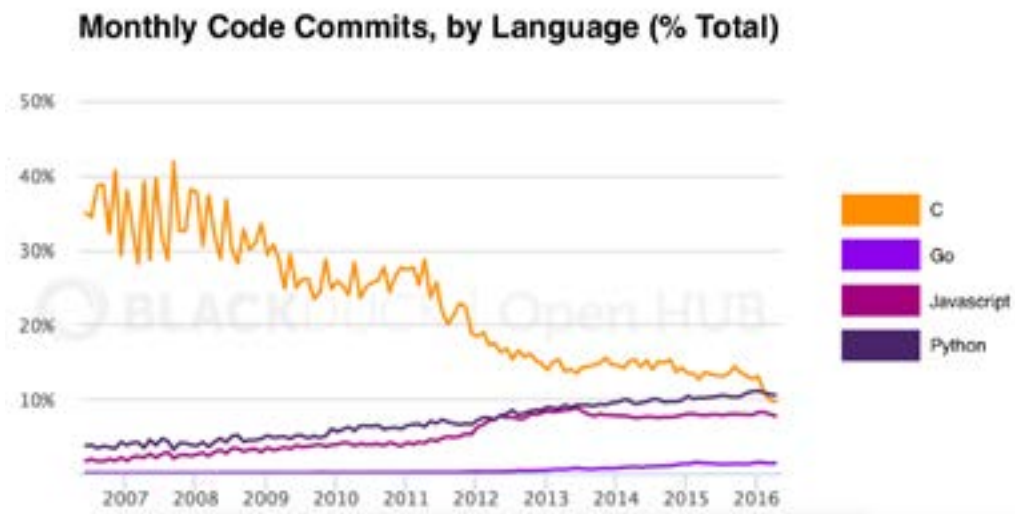
41

https://en.wikipedia.org/wiki/London_Underground

Digital infrastructure not only requires frequent maintenance and upkeep to be compatible with other software components, but its usage and adoption changes frequently as well. A bridge built in the middle of New York City will have fairly consistent and guaranteed usage, commensurate with the rise or decline of the city's

population. But a programming language or framework could be extremely popular for several years, then fall out of favor when something faster, more efficient, or simply trendier comes along.

For example, the graph below shows activity by source code developers using several different programming languages. The language C, one of the most fundamental and widely used languages, has dropped in market share as newer languages have entered the market. Python and JavaScript, two currently popular languages, are seeing a moderate rise over time. And Go, which was developed in 2007, has seen more activity in recent years.⁴²



42

Monthly code commits by language. Commits including multiple languages are counted once for each language. Black Duck's Open Hub pulls from over 650,000 open source projects. Data accessed May 20, 2016. <https://www.openhub.net/languages/compare>

Tim Hwang, who runs the Bay Area Infrastructure Observatory, which organizes group visits to physical infrastructure sites, remarked on the difference in a 2015 interview with *California Sunday Magazine*:

“A lot of [our] members work in tech, either on the web or on

*software. As a result, they work on things that don't last very long. Their approach is, 'We just hacked it, and we pushed it out live,' or 'We just released it, and we can work out bugs later.' A lot of infrastructure is built for 100 years. You can't have bugs. If you do, the building will fall down. You can't iterate it. It's a practice that exists outside of the members' day-to-day experience.*⁴³

43

<https://story.californiasunday.com/tim-hwang-infrastructure-tourist>

Because digital infrastructure changes so frequently, however, older projects have a harder time finding contributors, because many developers prefer to work on new and exciting projects. This phenomenon has been referred to as “magpie developer” syndrome, where developers are attracted to “new and shiny” things, instead of the technology that works best for them and their users.⁴⁴

44

<http://blog.codinghorror.com/the-magpie-developer/>

Digital infrastructure does not have a central organization to determine what gets built or used.

Finally, perhaps the most striking difference between digital and physical infrastructure, and one of the biggest challenges to its sustainability, is that there is no organizing body to determine what gets built or used in digital infrastructure.

Transportation, sewage and water projects are generally owned and managed by the government, whether federal, state or local. Communication and electric projects tend to be managed by private companies. In both situations, infrastructure projects are funded by a mix of private and public actors, either from the federal budget, private company financing, or metered user fees.

In a stable, developed country, we rarely think about whether or

how a road gets built or a building has electricity. Even for projects that are privately owned or funded, the federal government has a vested interest in guaranteeing that physical infrastructure gets built and maintained.

Digital infrastructure projects, on the other hand, are conceived of and built from the bottom up. It is akin to a group of citizens getting together and deciding they want to build a bridge or create their own sewage system. There is no authoritative body whose formal permission is required to create new digital infrastructure.

The Internet itself does have two major governing bodies that help set standards: the Internet Engineering Task Force and World Wide Web Consortium.

The Internet Engineering Task Force (IETF) helps develop and set voluntary standards for how information gets passed around the Internet. For example, they are the reason why URLs start with “HTTP”. They are also the reason why we have IP addresses—unique identifiers assigned to your computer when it is connected to a network. Originally a working group within the United States government in 1986, the IETF became an independent, international organization in 1993.⁴⁵ The IETF itself is run by volunteers, and there are no membership requirements: anyone from the public may join simply by declaring him- or herself a member.

45

https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force

The World Wide Web Consortium (W3C) helps set standards for the World Wide Web. It was founded by Tim Berners-Lee in 1994. The W3C tends to focus more exclusively on web pages and documents (they are, for example, the reason why web pages use HTML for basic formatting). They maintain the standards around the markup

language HTML and stylesheet formatting language CSS, two basic components of any web page. The W3C's membership is slightly more formalized, requiring an application and fee, and its members range from businesses to universities to individuals.

The IETF and W3C help manage standards around the most fundamental pieces of the Internet, but the next layer up—choices about which languages are used to build software, which frameworks to build them with, or which libraries to include—are entirely self-managed in the public domain. (Certainly, many proprietary software projects, particularly those with heavy regulation, such as aeronautics or health care, may have requirements on which tools are used. They may even build proprietary tools for their own use.)

With physical infrastructure, if the government builds a new bridge between San Francisco and Oakland, that bridge will certainly be used. Similarly, when the W3C wants to set a new standard, such as a new version of HTML, it is formally published and announced. For example, in 2014, the W3C announced HTML5, the first major revision of HTML since 1997, which had been in development for seven years.

By contrast, when a computer scientist wants to create a new programming language, he or she is free to publish it and it may or may not be adopted. The bar for adoption is even lower for frameworks or libraries: because they are easier to build, and easier for a user to learn and implement, these tools are iterated more frequently.

More importantly, nobody is forcing or even strongly encouraging anyone to use these projects. Some projects remain more academic than practical; others are ignored completely. It is difficult to predict

what gets used until others are actually using it.

Developers like to point to *usefulness* as an indicator of whether a new project gets adopted or not. New projects should make an improvement to an existing project, or solve a chronic problem, in order to be deemed useful and worthy of adoption. When developers are asked why their project got so popular, many of them will shrug and simply say, “It was the best thing out there.”

Not unlike technology startups, new digital infrastructure projects rely upon *network effects* for adoption. Getting a core group of developers excited, or a software company using the project, helps spread the word. A catchy name, branding, or website can add to the project’s novelty factor. A developer’s reputation within their respective community also helps determine whether a new project gets noticed.

However, in the end, a new digital infrastructure project can come from just about anywhere, which means each project is managed and sustained in very different ways.

How are digital infrastructure projects managed and supported?

We've established that digital infrastructure is as critical to modern society as physical infrastructure. Although digital infrastructure is not subject to the high costs and political obstacles of physical infrastructure, its decentralized nature also makes it harder to pin down. Without a central governing body, how do open source projects find the support they need?

In short, the answer is different for every project. However, there are several places where projects might originate: within a company, as a new business, or from an individual or community of developers.

Within a company

Sometimes, the project starts within a company. Here are a few examples that demonstrate the different ways in which an open source project might be supported by a company's resources:

*Go, the new programming language previously mentioned, was developed at Google in 2007 by engineers Robert Griesemer, Rob Pike, and Ken Thompson, who created Go as an experiment. Go is open source and accepts contributions from the broader community. However, its core maintainers are employed full-time by Google to work on the language.*⁴⁶

*React is a new JavaScript library that is growing in popularity. React was created by Jordan Walke, a software engineer at Facebook, for internal use on Facebook's news feed. An employee at Instagram (which is owned by Facebook) wanted to use React, too, and eventually React was open sourced, two years after its initial development.*⁴⁷ Facebook

46

[https://en.wikipedia.org/wiki/Go_\(programming_language\)](https://en.wikipedia.org/wiki/Go_(programming_language))

47

<https://www.quora.com/How-was-the-idea-to-develop-React-conceived-and-how-many-people-worked-on-developing-it-and-implementing-it-at-Facebook/answer/Bill-Fisher-17>

dedicated a team of engineers to help maintain the project, but React also accepts contributions from the public developer community.⁴⁸

48

[https://en.wikipedia.org/wiki/React_\(JavaScript_library\)](https://en.wikipedia.org/wiki/React_(JavaScript_library))

***Swift, the programming language used for iOS, OS X, and other Apple projects,** is an example of a project that was only recently open sourced. Swift was developed internally by Apple for four years and released as a proprietary language in 2014. Developers could use Swift to write software for Apple devices, but not contribute to the language's core development. In 2015, Swift was open sourced under the Apache License 2.0.⁴⁹*

49

[https://en.wikipedia.org/wiki/Swift_\(programming_language\)](https://en.wikipedia.org/wiki/Swift_(programming_language))

The incentives for a company to maintain an open source project are numerous. Opening up a project to the public can mean less work for the company, which is essentially crowdsourcing improvements. It builds goodwill and awareness among developers, who might then be incentivized to use other company resources to build things. Having an active community of developers creates a recruiting pipeline for talent. And sometimes, open sourcing a project helps a company strengthen their user base and brand, or even drown out competition. The more market share a company can capture, even through tools it gives away, the more influential it becomes. This is not dissimilar to the “loss leader” concept of business.

Even if a project is created internally, if it is open sourced, that project is free to use or modify according to the terms of an open source license, and is not considered company intellectual property in the traditional sense. Many company projects use standard open source licenses that are considered acceptable by the broader developer community, such as Apache License 2.0 or BSD. However, in some cases, companies add their own clauses. React, for example, has an additional clause that could potentially cause patent claim conflicts with React users.⁵⁰ As a result, some companies and

50

<https://github.com/facebook/react/blob/master/PATENTS>

individuals are reluctant to use React, and the decision is frequently portrayed as in conflict with open source principles.

As a new business

Some infrastructure projects take a traditional startup path, including venture funding. A couple of examples are as follows:

Docker, perhaps the best-known contemporary example, helps software applications run inside containers. (Containers provide a clean, tidy environment for software applications that make them easier to run anywhere). Docker started as an internal project within dotCloud, a platform-as-a-service company, but became so popular that the founders decided to make Docker the main focus of the company. The Docker project was open sourced in 2013. Docker has raised \$180M with an estimated valuation of over \$1B.⁵¹ Their business model is based on support, private plans, and services. Docker's 2014 revenue was less than \$10 million.⁵²

Npm is a package manager to help Node.js developers share and manage their projects, released in 2010. Npm raised nearly \$11M in funding since 2014 from True Ventures and Bessemer Ventures, among others. Their business model focuses on paid features that support privacy and security.

Meteor is a JavaScript framework that was first released in 2012. It was incubated by Y Combinator, a prestigious startup accelerator that also incubated companies like AirBnB and Dropbox. Meteor has received over \$30M in funding to date from firms including Andreessen Horowitz and Matrix Partners.⁵³ Meteor's business model focuses on an enterprise platform called Galaxy, released in October 2015, for operating and managing Meteor applications.⁵⁴

51

<http://venturebeat.com/2015/06/13/docker-now-valued-at-1b-paid-some-one-799-for-its-logo-on-99designs/>

52

<http://www.bloomberg.com/news/articles/2015-04-14/docker-said-to-join-1-billion-valuation-club-with-new-funding>

53

<https://www.crunchbase.com/organization/meteor>

54

<http://info.meteor.com/blog/announcing-meteor-galaxy>

The venture funding approach is relatively new, and growing rapidly. Lightspeed Venture Partners found that from 2010-2015, venture firms invested over \$4B in open source companies, a tenfold increase over the five previous years.⁵⁵

Using venture capital to support open source projects has been met with skepticism from developers (and even some venture capitalists themselves), due to lack of clear business models and questionable revenue to justify valuations. Steve Klabnik, a maintainer for the language Rust, explains venture capital's sudden interest in funding open source:

“I’m a VC. I need a large number of companies to exist to make my money....I need costs to be low and profits to be high. I need a healthy open source ecosystem to make this happen. So what do I do?...VCs are realizing this story, and are starting to invest in infrastructure. [...]

*In many ways, the open source stuff is a loss leader, so that you get hooked...and then use it for everything, even your closed source code. This is a great business strategy, but it also places GitHub at the center of this new universe. So for similar reasons, a16z needs GitHub to be awesome to bootstrap every open source ecosystem that will exist into the future....And a16z has the money to ‘throw away’ on something they won’t get a direct return out of, because they’re smart enough to invest some of their fund in ecosystem development.*⁵⁶

GitHub, created in 2008, is a platform for code, available publicly or privately in an easy-to-read environment. It hosts many popular open source projects and, most importantly, has become the cultural

55

<http://venturebeat.com/2015/12/06/its-actually-open-source-software-thats-eating-the-world/>

56

<http://words.steveklabnik.com/is-npm-worth-26mm>

epicenter for open source’s explosive growth (to be discussed later in this report).

GitHub did not take any venture capital until 2012, four years after its founding. Before then, GitHub was a profitable company. Since 2012, GitHub has taken \$350M in total venture capital funding.⁵⁷ Andreessen Horowitz (or “a16z”), the \$4B venture capital firm who provided most of the capital in their first \$100M round, stated it was the largest investment they had ever made at the time.⁵⁸

57

<https://www.crunchbase.com/organization/github>

58

<http://www.wsj.com/articles/SB10001424052702303292204577517111643094308>

Steve Klabnik’s thesis, in other words, is that venture capital firms who invest in open source infrastructure promote these platforms as a “loss leader,” even when there is no direct business model or profitability to be had, because it grows the entire ecosystem. The more resources GitHub has, the more open source thrives. The more open source thrives, the more startups thrive. If nothing else, venture capital’s interest in open source, especially given the lack of clear financial return, validates the critical role open source plays in the broader startup ecosystem.

(As an aside, it is important to mention that GitHub, the platform itself, is not an open source project, and therefore is not an example of venture capital directly funding open source. GitHub is a closed source platform that hosts open source projects. This is a controversial topic for some open source contributors.)

By individuals or a group of individuals

Finally, many digital infrastructure projects are developed and maintained entirely by independent developers, or a community of developers. A few examples are as follows:

Python, a programming language, was developed and published by computer scientist Guido van Rossum in 1991. Van Rossum claimed he “was looking for a ‘hobby’ programming project that would keep me occupied during the week around Christmas.”⁵⁹ The project took off, and Python is now considered to be one of the most popular programming languages today.⁶⁰ Van Rossum remains the principal author of Python (also known as a benevolent dictator for life, or BDFL, among developers) and is currently employed by Dropbox, whose software relies heavily on Python.⁶¹ Python is partially managed by the Python Software Foundation, created in 2001, which has a number of corporate sponsors, including Intel, HP, and Google.

RubyGems is a package manager that helps distribute programs and libraries associated with the Ruby programming language. It is a critical piece of infrastructure for any Ruby developer. Examples of websites that use Ruby are Hulu, AirBnB and Bloomberg.⁶² RubyGems was created in 2003 and is managed by a community of developers. Some development work is supported by Ruby Together, a foundation that accepts donations from companies and individuals.

Twisted, a Python library, was authored by a developer named Glyph Lefkowitz in 2002. Since then, it has achieved widespread usage among individuals and organizations, including Lucasfilm and NASA.⁶³ Twisted continues to be run by a group of volunteers. It is supported by corporate and individual donations; Lefkowitz remains the lead architect and offers consulting services for income.⁶⁴

59

<https://www.python.org/doc/essays/foreword/>

60

<http://blog.codeeval.com/codeevalblog/2015#.VjvKZhNViko=>

61

https://en.wikipedia.org/wiki/Guido_van_Rossum

62

<http://skillcrush.com/2015/02/02/37-rails-sites/>

63

<https://twistedmatrix.com/trac/wiki/SuccessStories>

64

<https://twistedmatrix.com/glyph/>

As these examples demonstrate, open source projects can come from just about anywhere. This is, generally, considered to be a good thing. It means that *useful* projects are more likely to succeed, avoiding both the vacuous hype associated with startups, and the bureaucracy associated with government. Digital infrastructure’s decentralized nature also reinforces the open and democratic

principles of the Internet, where anybody could theoretically create the next big project, whether a company or individual.

On the other hand, many useful projects will come from independent developers who suddenly find themselves at the helm of a successful project, facing critical decisions about its future. A 2015 study by the Federal University of Minas Gerais in Brazil looked at 133 of the most actively used projects hosted on GitHub, across programming languages, and found that 64%, or nearly two-thirds, relied upon just one or two developers to survive.⁶⁵ Although there may be a long tail of casual or infrequent contributors, for many projects, the major responsibilities of project management fall on just a few people.

65

<https://peerj.com/preprints/1233.pdf>

Coordinating international communities of opinionated contributors and managing the expectations of Fortune 500 companies who use your project are challenging tasks for anyone. It is truly impressive how much has already been accomplished in this manner. These tasks are especially difficult when developers lack clear role models or institutional support for this work. In interviews for this report, many developers privately lamented that they have no idea who to ask for help and would “rather just code.”

Why do they keep doing it? The remainder of this paper will focus on how and why open source contributors maintain projects of massive scale and impact, and why it matters to all of us.

Why do people keep contributing to these projects, when they're not getting paid for it?

Many digital infrastructure projects are maintained by individual contributors or a community of contributors. In most cases, these contributors are not being paid directly to work on the project. Instead, they contribute for reasons that are unique to open source communities, including building reputation and a public service mindset. This section will explore some of those motivations in greater detail.

Contributing to open source builds one's reputation.

Building one's reputation is perhaps the most practical reason why someone would want to contribute to an open source project. For developers, technical writers, or others, these projects help them prove themselves in public, giving them a chance to be part of something big and useful.

Google runs a program called Google Summer of Code, which provides a summer stipend to student developers to contribute to popular open source projects. The program works well because the developers are students, new to the field of computer science, and eager to show off their skills.

Developers, in particular, leverage open source contributions to build a portfolio of their work. In addition, by providing input on popular projects with active communities, a developer has a chance

to build his or her reputation by making him or herself “known.” GitHub, the website previously mentioned, is a popular platform for collaborating on code. When a developer makes a contribution to a public software project, those contributions appear in his or her profile. A developer’s GitHub profile can serve as a portfolio for software companies, but only contributions to public (i.e., open source) projects are visible to anyone.

However, reputation-based motivations also come with risks, especially among junior developers. A developer early in his or her career may contribute to an open source project for the sole purpose of getting hired, then stop contributing once this goal has been achieved. In addition, developers who are solely interested in building their portfolio may make lower quality contributions to the project that do not get accepted or even slow down the development process. Finally, if the purpose of making a public contribution is to build one’s reputation, a developer will be motivated to only contribute to popular or attractive projects (an extension of the “magpie developer” phenomenon mentioned earlier), which means that older projects struggle to find new contributors.

The project became unexpectedly popular, and the maintainer feels obligated to support it.

A popular open source project can create dependencies for other companies, individuals or organizations. In other words, the code is being used in live software, written and deployed by other people, that could serve any number of purposes, whether online shopping or health care. This complex set of dependencies (many of which are not visible even to the project author, since they do not have clear

user data) can make a maintainer feel ethically obligated to continue supporting it.

Arash Payan, the developer of Appirater mentioned in the beginning of this paper, released his project in 2009. Of his decision to continue maintaining the project, Payan says:

*“It's not terribly exciting stuff, but there are so many people out there that use (depend, even?) on the project for their apps, that I feel obligated to be a good steward of it. Personally, I've moved on from iOS, so maintaining an iOS library isn't exactly my first choice for a side project.”*⁶⁶

66

Email interview with Arash Payan

Payan estimates the project only takes roughly 1-2 hours per month to keep up-to-date, so he doesn't mind the work.

Some projects become unexpectedly popular but take more time to support. Andrey Petrov is an independent developer who wrote a Python library called `urllib3`. He released it in 2008 as a significant improvement to the existing standard library, and it became popular among Python developers. Today, every Python user depends on it.⁶⁷

67

<https://medium.com/@shazow/urllib3-stripe-and-open-source-grants-edb-9c0e46e82>

Andrey made the project open source in hopes that other people would help support its continued development and upkeep. Andrey is a freelance developer—although he enjoys maintaining `urllib3`, he can only justify doing so in his free time, since he is not paid for his work. Cory Benfield, who is employed by Hewlett Packard Enterprise to help maintain critical Python libraries (which HPE uses and depends on), now works on `urllib3` as part of his job. Cory's arrangement has reduced some of the burden.

The project is a labor of love.

Eric Holscher is a creator of Read the Docs, which hosts software documentation. Documentation is the equivalent of an instruction manual. Just as one might need an instruction manual to figure out how to put a piece of furniture together, developers need documentation to figure out how to implement a project. Without proper documentation, it would be difficult for a developer to know how to get started.

Read the Docs provides documentation for 18,000 software projects, including enterprise customers, and serves over 15 million page views per month.⁶⁸ Although they make some money from large business clients, Read the Docs is still mostly funded by donations from its users. A company called Rackspace sponsors their server costs.

Eric and his cofounder, Anthony Johnson, maintain the project, and do not see steady income from it, despite working on the project full-time. A \$48,000 one-time grant from the Mozilla Foundation in December 2015 will help cover their work for a short time.⁶⁹ They are currently experimenting with an advertising model (that does not involve tracking their users) to reach sustainability.⁷⁰

Eric notes that the difficulties lie not just in new development work, but non-coding functions like customer support, for which one of the maintainers must be on call every weekend in case of an emergency. When Eric explained why he continues to support the project, he called it a “labor of love”:

68

<https://readthedocs.org/sustainability/>

69

<https://blog.mozilla.org/blog/2015/12/10/mozilla-open-source-support-first-awards-made/>

70


<http://blog.readthedocs.com/ads-on-read-the-docs/>

“Either humans are irrational or they don’t just follow money. Clearly there’s another motivation for me here. It’s a labor of love. I could close this project tomorrow and be done with it if I wanted to, but I’ve been doing it for 5 years and I don’t want to see that happen.”⁷¹


71

—
Skype interview with
Eric Holscher

Eric is motivated to work on Read the Docs because he sees the tangible value it creates for others. For many project maintainers, impact is a primary motivator, because they see how their direct efforts positively affect other people’s lives. In this sense, open source work shares many similarities to the nonprofit sector. Much like the nonprofit sector, however, this “labor of love” mentality can make it harder for open source communities to discuss the elephant in the room: how to sustain projects that require more resources and attention than current contributors can offer.



Challenges Facing Digital Infrastructure



"Open source has been such an incredible force for quality and community exactly because it's not been defined in market terms. In market terms, most open source projects should never have had a chance."

– David Heinemeier Hansson, Ruby on Rails

Open source's complicated relationship with money

Money has been a taboo topic for open source projects since the early days of the free software movement, which arose in direct response to the practice of commercial, proprietary software.

In the context of the free software movement, the aversion to money is certainly understandable. Money is what commercialized software in the 1980s, and it took decades to reverse this mentality and promote the benefits of building software that is free to use, distribute, and modify. Although we take free software for granted today, in the 1980s it was truly a counterculture and revolutionary mindset.

Yet even within open source communities, there is a pervasive belief that money has a corrupting influence on open source. It is indeed remarkable how much has been created entirely through labors of love. These days, software development is considered a lucrative field, with coding schools luring prospective students with the promise of making a six-figure entry-level salary. By contrast, there is something pure and admirable about creating a software project simply for the joy of it.

On a more practical level, open source projects traditionally arise from a real and identifiable need. Someone decides that a project could be done better, so they fork the project, make improvements, then release it for consumption. Pragmatism is core to open source's culture, as evidenced by its strategic break from the free software movement in the late 1990s. Some open source contributors fear,

perhaps justifiably, that money will introduce bloat into the system, with developers creating new projects simply to get funding, rather than because the solution is needed.

David Heinemeier Hansson (also known as DHH), who created the popular software framework Ruby on Rails, warned in 2013 against mixing open source with money:

“Open source has been such an incredible force for quality and community exactly because it's not been defined in market terms. In market terms, most open source projects should never have had a chance.

Take Ruby on Rails. [...] That's a monumental achievement of humanity! Thousands, collaborating for a decade, to produce an astoundingly accomplished framework and ecosystem available to anyone at the cost of zero. Take a second to ponder the magnitude of that success. Not just for Rails, of course, but for many other, and larger, open source projects out there with an even longer lineage and success.

It's against this fantastic success of social norms that we should be extraordinary [sic] careful before we let market norms corrupt the ecosystem.⁷²

Structurally, open source's greatest advantage —its penchant for democracy— is also its weakness. Many open source projects are nothing more than a public code repository to which a group of people contribute regularly: the equivalent of an unofficial social club on a college campus. There is no legal structure and there are no clear owners or leaders. “Maintainers,” or the primary

72

<http://david.heinemeierhansson.com/2013/the-perils-of-mixing-open-source-and-money.html>

contributors, often emerge *de facto*, based on who authored the project or put in significant time or effort. Even then, however, some projects are reluctant to introduce hierarchy by clearly favoring one contributor over another.

In April of 2008, Jeff Atwood, an aforementioned prominent .NET developer, announced he was donating \$5,000 towards an open source project, ScrewTurn Wiki. ScrewTurn Wiki is a wiki project developed by Dario Solara, another .NET developer, and maintained by volunteers. Atwood told Dario that the grant would be “no strings attached;” Solara could use the money as he saw fit towards the project.

Several months later, Atwood followed up with Solara to ask how he decided to spend the donation. Solara replied that the grant money was “*still untouched. It’s not easy to use it...What would you suggest?*” Atwood wrote that he was “crushingly disappointed” by the response.⁷³ The decentralized nature of open source has made it what it is: crowdsourced software that anyone can build, share and contribute to. But when it comes to discussing organizational needs or sustainability, it can be difficult to make authoritative decisions.

These transitions to long-term sustainability can be drawn out and painful. One of the more prominent examples is the Linux kernel, an open source project used in many operating systems worldwide, including Android and Chrome OS. It was created in 1991 by computer science student Linus Torvalds.

As the Linux kernel grew in popularity, Linus was reluctant to discuss how to scale development of the project, preferring to manage everything himself. Project maintainers grew restless and

73

<http://blog.codinghorror.com/is-money-useless-to-open-source-projects/>

even angry at Torvalds, sparking “really big fights,” according to Torvalds. The disputes peaked in 2002 with discussions of a potential schism.

Torvalds attributed the internal conflict to a lack of organization, rather than to any technical issues:

“We had really big fights back in 2002 or so where I was dropping patches left and right, and things really weren't working. It was very painful for everybody, and very much for me, too. Nobody really likes criticism, and there was a lot of flaming going around—and because it wasn't a strictly technical problem, you couldn't point to a patch and say, ‘hey, look, that patch improves timings by 15%’ or anything like that: there was no technical solution. The solution ended up being better tools, and a work flow [sic] that allowed much more distributed management.”⁷⁴

74

<http://www.datamation.com/open-source/linus-torvalds-and-others-on-community-burnout-1.html>

The Linux Foundation was created in 2007 to help protect and maintain Linux and its associated projects. Torvalds does not run the Linux Foundation himself, preferring instead to receive a steady salary as a “Linux Fellow” and work on his projects as an engineer.

While open source software is admirably rooted in a culture of volunteerism and collaboration relatively untouched by extrinsic motives, the reality is that our economy and society, from multimillion dollar companies to government websites, depends on open source.

Overall, this is probably a positive development for society. It means that software is no longer strictly relegated to private, proprietary

development, as it had been for decades. The fact that the United States government, or a social network website with billions of users, incorporates community-built software, paints an optimistic future for democracy.

In addition, many projects function well on a community basis if they are on the extremes of size: that is, either small projects that do not require significant maintenance (as in the example of Arash Payan and Appirater), or very large projects that have found significant corporate support (as in the example of Linux).

However, many projects are trapped somewhere in the middle: large enough to require significant maintenance, but not quite so large that corporations are clamoring to offer support. These are the stories that go unnoticed and untold. From both sides, these maintainers are told they are the problem: Small project maintainers think mid-sized maintainers should just learn to cope, and large project maintainers think if the project were “good enough,” institutional support would have already come to them.

There are also political concerns around financial support that make it harder to find a reliable source of funding. A single company may not want to sponsor development work that also benefits their competitor, who paid nothing. A private benefactor may want special privileges that threaten the neutrality of a project. (For example, for security-related projects, privileged disclosure of vulnerabilities—paying for special knowledge about security vulnerabilities instead of exposing those vulnerabilities to the public—is a controversial request.) And governments may have political reasons to sponsor the development of a particular project, or ask for special favors

such as “backdoors” (a secret way of bypassing security authentication), even if that project is used internationally.

The recent legal disputes between the FBI and Apple help underscore the tension between technology and government, even beyond open source projects. The FBI has repeatedly, through court orders, requested Apple’s assistance in unlocking iPhones to help resolve criminal investigations. Apple has repeatedly denied these requests. In February 2016, the FBI requested Apple’s assistance in unlocking the iPhone belonging to one of the shooters in a recent terrorist attack in San Bernardino, California. Apple again denied the request, posting a public customer letter on its website, which stated that:

“While we believe the FBI’s intentions are good, it would be wrong for the government to force us to build a backdoor into our products. And ultimately, we fear that this demand would undermine the very freedoms and liberty our government is meant to protect.”⁷⁵

75

<http://www.apple.com/customer-letter/>

In March 2016, the FBI found a third party to help it unlock the iPhone and dropped the legal case.

One of open source’s greatest strengths is that the code is considered a public good, and many projects take governance seriously. It is personally important to many project maintainers that no individual party control something that the public uses and benefits from. However, this commitment to neutrality can come at a price, when many resources available to software developers today (such as venture capital or corporate donations) are based on expectations of

influence or financial return.

Open source software is being created and used at a rate never seen before. Many open source projects are experiencing a difficult transition from selfless creative pursuit to critical public infrastructure. These increasing dependencies mean we have a shared responsibility to ensure that these projects find the support they need.

Why digital infrastructure support problems are accelerating

Open source is quickly becoming a standard for digital infrastructure projects, and in software development overall, due to the benefits cited earlier in this paper. Black Duck, a company that helps clients manage open source software, runs an annual survey that asks companies about their open source use. (This survey is one of the few open source data projects in existence.) According to their 2015 survey, 78% of the 1,300 companies surveyed said their software created for customers was built on open source, nearly double that of 2010.⁷⁶

Open source has seen massive growth in popularity in the last five years, not just because of the obvious benefits to developer and consumer, but also due to new tools that make it easier to collaborate on software. In order to understand why digital infrastructure faces growing support problems, we must understand how open source software development is proliferating.

GitHub as a standardized place to collaborate on code

The role of GitHub in bringing open source to a mainstream audience cannot be overemphasized. Although open source has existed for over thirty years, until 2008, contributing to an open source project was not so easy. A developer would have to figure out who the maintainer was, find a way to contact them, and propose changes using whichever format that project maintainer preferred (for example, a mailing list or message board). GitHub standardized

76

<https://www.blackducksoftware.com/future-of-open-source>

these methods of communication: maintainers are transparently listed on a project's page, and discussion of proposed changes takes place on the GitHub platform.

GitHub created vocabulary that is now standard among open source contributors, such as the "pull request" (where a developer submits changes to a project for review) and repurposing the term "fork" (historically, creating a copy of a project and modifying it into a new project). Prior to GitHub, forking a project meant there were irreconcilable differences over the direction a project should take. Forking was considered a serious move: if a group of developers forked a project, it meant the project was splitting into ideological factions. Forking was also used to develop a new project that might have a markedly different purpose from the original project.

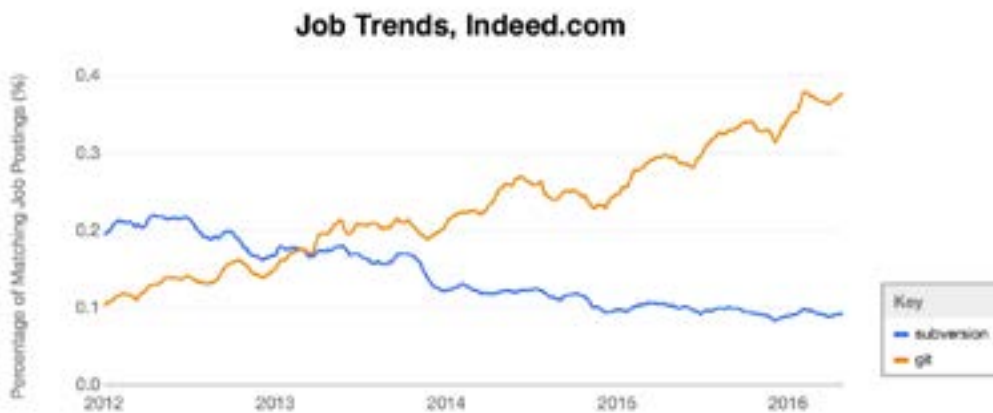
This type of "project fork" still exists today, but GitHub decided to use the term "fork" to encourage more activity on their platform. A GitHub fork, unlike a project fork, means temporarily copying a project, making changes, and usually merging it back into the project. Forking as an everyday practice on GitHub's platform has added a positive, lightweight connotation to the term: a sign of taking one person's idea and making it better.

GitHub also helped standardize the use of a version control system called Git. Version control systems keep track of everybody's work on a particular piece of code. For example, if Developer 1 and Developer 2 are fixing different parts of the same code at the same time, logging each change in a version control system ensures their changes don't conflict with one another.

There are several options for version control systems, including

Apache Subversion and Concurrent Versions System (CVS). Git used to be a fairly unknown version control system. In 2010, Subversion was used in 60% of software projects, whereas Git was used in just 11%.⁷⁷

Linus Torvalds, the developer behind Linux, designed Git in 2005 as a better and faster way to manage multiple contributions from many people. Git was markedly different from earlier version control systems and therefore not so easy to pick up, but its decentralized workflow solved a real problem for developers.



79

GitHub provided an intuitive user interface for open source projects that used Git, thus making it easier for developers to learn. The more developers used GitHub, the more they wanted to keep using Git. Today, in 2016, Git is used in 38% of software projects, while Subversion's share has dropped to 47%.⁷⁸ Although Subversion is still the most popular version control system today, its use is declining.

77

<http://redmonk.com/so-grady/2013/12/19/dvcs-and-git-2013/#ixzz2qyfVpSR9>

79

Job postings requiring knowledge of Git vs. Subversion, via Indeed.com. Data accessed June 7, 2016. <http://www.indeed.com/jobtrends>

78

As of January 6, 2016. <https://www.openhub.net/repositories/compare>

The widespread adoption of Git makes it much easier for a developer to join any open source project on GitHub, because the method of making changes, and communicating those changes, is the same across projects. Learning how to contribute to one project gives someone the ability to contribute to hundreds of others. This was not the case before GitHub, where different version control systems were used for different projects.

Finally, GitHub provided a place for developers to talk to each other in a public setting for social reasons, not just code collaboration. GitHub has become a de facto community of sorts for developers, who use it to communicate with each other and show off their work. Developers now have an opportunity to demonstrate their leadership and portfolio of work in a way they could not before.

GitHub's usage reflects its meteoric rise. In 2011, there were only 2 million repositories.⁸⁰ Today, GitHub has 14 million users and over 35 million repositories.⁸¹ (Note that this includes forked repositories; the unique repository count is probably closer to 17 million.) GitHub's Brian Doll noted that the first million repositories took nearly 4 years to create; getting from nine to ten million took just 48 days.⁸²

By contrast, SourceForge, the most popular platform for hosting open source code before GitHub, had 150,000 projects in 2008. An estimated 18,000 of those projects were active.⁸³

80

<https://github.com/blog/841-those-are-some-big-numbers>

81

<https://en.wikipedia.org/wiki/GitHub>

82

<https://github.com/blog/1724-10-million-repositories>

83

<http://dirkriehle.com/publications/2008-2/the-total-growth-of-open-source/>

Stack Overflow as a standardized place to get help with code

Another important tool is Stack Overflow, a popular Q&A website for programmers, also created in 2008 by Jeff Atwood, the aforementioned programmer and blogger, and Joel Spolsky. As of April 2014, Stack Overflow had over 4 million registered users and over 11 million questions answered (note that one does not need to be registered to view questions or their answers).⁸⁴ Stack Overflow has become a *de facto* support platform for developers to ask questions about coding, find answers to specific code problems, or just get advice on the best way to build a certain piece of software. It can be thought of as crowdsourced “customer support” for developers around the world. While Stack Overflow is not a place to write live code, it is a critical collaboration tool for the individual developer, making it easier to solve problems and code more efficiently. This means any one individual developer is capable of achieving more in a shorter period of time, increasing overall output. Stack Overflow has also helped people learn new coding concepts (or even get started with coding itself), making coding easier and more accessible to all.

84

https://en.wikipedia.org/wiki/Stack_Overflow

Macro trends in a rapidly changing landscape

The outsized popularity of open source has led to significant changes in how today’s developer talks, thinks about, and collaborates on software.

Firstly, licensing expectations and requirements have changed to reflect a world that embraces open source as the standard, not the exception: a triumph over the proprietary world of the 1980s. Both GitHub and Stack Overflow’s policies reflect this.

From the beginning, Stack Overflow used a Creative Commons license called CC-BY-SA for all content posted to the website. This license was limiting, however, because it required that users provide attribution when using others' code, as well as distribute contributions under the same license.⁸⁵ While many ignored or were not aware of this license, it made it difficult for developers under stricter constraints (such as in a corporate environment) to use Stack Overflow. If they posted a question asking for help with their code, and a stranger fixed it, legally, they would have had to attribute the code to that person.

85

<https://creativecommons.org/licenses/by-sa/2.0/>

As a result, Stack Overflow announced an intent to move all new code contributions to the MIT License, which is an open source license with fewer restrictions.⁸⁶ As of April 2016, they are still actively discussing and soliciting feedback from the community on the best way to implement more permissiveness.⁸⁷ This move is a nod to both Stack Overflow's popularity and the proliferation of open source at large. That a developer working at a big software company could legally include a complete stranger's code in a product they charge for is an accomplishment for open source, indeed.

86

<http://meta.stackexchange.com/questions/271080/the-mit-license-clarity-on-using-code-on-stack-overflow-and-stack-exchange>

87

<http://meta.stackexchange.com/questions/272956/a-new-code-license-the-mit-this-time-with-attribution-required>

GitHub, by contrast, initially avoided providing default licensing for projects posted to its platform, perhaps fearing it would slow user adoption and growth.⁸⁸ Projects posted to GitHub, then, grant the right to view and fork the project, but are otherwise protected under copyright, unless the developer specifies an open source license.

88

<http://www.infoworld.com/article/2615869/open-source-software/github-needs-to-take-open-source-seriously.html>

In 2013, facing public concerns, GitHub finally decided to take a stronger stance on licensing, including the creation and promotion of a microsite, choosealicense.com, to help users pick a license for

their project. They also now encourage their users to choose a license from a list of options when creating a new repository.⁸⁹

What's interesting, however, is that many developers were either not aware that their "open source" projects were not legally protected, or didn't care. An informal 2013 study by the Software Freedom Law Center of 1.6 million GitHub repositories revealed that only 15% had specified a license.⁹⁰ Conversations with developers for this report suggested that many didn't care to put up a license, or figured that if someone asked, they could just add one later.

This lack of interest in licensing led James Governor, cofounder of developer analyst firm Red Monk, to observe in 2012 that "*younger devs today are about POSS - Post open source software. Fuck the license and governance, just commit to Github*".⁹¹ In other words, defaulting to open information is so culturally obvious today that developers don't see themselves as doing something differently anymore, the way the political free software rebels did in the 1980s. This shift in values, while inspiring on a macro level, could lead to legal complications for individuals as their projects grow in popularity or are used for commercial purposes.

But by making it so easy and standardized to collaborate on code together, open source is also grappling with a perverse set of externalities.

Open source made coding easier and more accessible to the world. This increased accessibility, in turn, has created a new class of developers who are less experienced, but who know how to utilize others' prefabricated components to build what they need.

89

<http://www.infoworld.com/article/2611422/open-source-software/github-finally-takes-open-source-licenses-seriously.html>

90

http://www.theregister.co.uk/2013/04/18/github_licensing_study/

91

<https://twitter.com/monkchips/status/247584170967175169>

In 2012, Jeff Atwood, the cofounder of Stack Overflow, wrote a tongue-in-cheek blog post called “Please Don’t Learn How to Code”, lamenting the trendiness of coding bootcamps and schools. While Atwood commended the desire of nontechnical people to understand code on a conceptual level, he warned against assuming that *“adding naive, novice, not-even-sure-they-like-this-whole-programming-thing coders to the workforce is a net positive for the world.”*⁹²

92

<http://blog.codinghorror.com/please-dont-learn-to-code/>

Under these circumstances, the open source development model looks different than it did before. Prior to GitHub’s rise, because there were fewer open source projects, developers were a smaller but on the whole more experienced group, and those who used shared code were likely also the people contributing back.

Today, the hypergrowth of coding literacy means many inexperienced developers are flooding the market. These newer developers borrow shared code to write what they need, but they are less capable of making substantial contributions back to those projects. Many are also accustomed to thinking of themselves as “users” of open source projects, rather than members of a community. Because open source tools are more standardized and easy to use, it’s much easier these days for someone to pop into a GitHub forum and make a rude comment or demanding request, which burdens and exasperates project maintainers.

These changing demographics have also led to a much more fragmented system of software, with many developers releasing new projects and creating a confusing web of dependencies. Drew Hamlett, who calls himself a “recovering magpie developer,” wrote a popular post in January 2016 called “The Sad State of Web Development,” about how web development has changed, referring specifically to the Node.js ecosystem:

“The people who have stayed in the Node community have undoubtedly created the most over engineered eco system [sic] that has ever appeared. No one can create a library that does anything. Every project that creeps up is even more ambitious than the next....No one will build something that actually does anything. I just don't understand. The only thing I can think, is people are just constantly re writing Node.js apps over and over.”⁹³

There are so many projects being written and released today that it is simply not feasible for each one to grow a large, sustainable community with regular contributors who passionately discuss changes over extensive mailing list discussions. Instead, many projects will be maintained by just one or two people. But demand for those projects by users might still outpace the work that is required to maintain it.

GitHub made it easy to create and contribute to new projects. This was a blessing for the open source ecosystem, because projects develop more rapidly, but it can be a curse to any one project maintainer, with more people easily reporting issues and requesting new features, without actually contributing back themselves. These shallow interactions only create more work for the maintainers, who are expected to address a growing volume of requests.

It would not be unreasonable to suggest that a “post-open source” world carries implications not just for licensing, as James Governor originally intended with his comment, but for the process of development itself.

93

<https://medium.com/@wob/the-sad-state-of-web-development-1603a861d-29f#4431c2nv1>

Noah Kantrowitz, a longtime Python developer and member of the Python Software Foundation, summarized this shift in a widely cited blog post:

In the early days of the open source movement there were relatively few projects and in general most people using a project were also contributing back to it in some way. Both of these have changed by likely uncountable orders of magnitude.

[...] As we have moved to more and more niche tools, it becomes harder to justify the time investment to become a contributor. ‘Scratching your own itch’ is still a powerful motivator, but that alone is difficult to build an ecosystem on.

*The other problem is the growing imbalance between producers and consumers. In the past, these were roughly in balance. Everyone put time and effort in to the Commons and everyone reaped the benefits. These days, very few people put in that effort and the vast majority simply benefit from those that do. This imbalance has become so ingrained that for a company to re-pay (in either time or money) even a small fraction of the value they derive from the Commons is almost unthinkable.*⁹⁴

94

<https://coderanger.net/funding-foss/>

This is not to say that big open source projects with strong contributor communities do not exist anymore. (Node.js, which will be discussed later in this paper, is an example of a project that has achieved this status.) It is that in addition to these successes, there is a new class of projects today that is underserved by open source’s current norms and expectations, and that the behavior deriving from these new norms has affected even longer-running, bigger projects.

Hynek Schlawack, a Python Software Foundation fellow and contributor to Python infrastructure projects, frets about a future with a wider demand base but only a handful of keystone contributors:

“What frustrates me most is that we have an all-time high of Python developers and an all-time low on high quality contributions.[...] As soon as pivotal developers like Armin Ronacher slow down their churn, the whole community feels it immediately. The moment Paul Kehrer stops working on PyCA we’re screwed. If Hawkowl stops porting, Twisted will never be on Python 3 and git.

So we’re bleeding due to people who cause more work than they provide. [...] Right now everyone is benefitting from what has been built but due to lack of funding and contributions it’s deteriorating. I find that worrying, because Python might be super popular right now but once the consequences hit us, the opportunists will leave as fast as they arrived.”⁹⁵

95

Email interview with Hynek Schlawack

Open source has only been popular among mainstream developers for perhaps five years; its long-term sustainability is rarely discussed, or even acknowledged, by the broader software community. With the explosion of new developers using, but not giving back to, shared code, we are building palaces on top of crumbling infrastructure.

The hidden costs of ignoring infrastructure

As we've seen, digital infrastructure is a critical part of today's world. Our society is built on software, and that software is increasingly built on infrastructure that uses open source methodology. By not taking steps to understand and support our digital infrastructure, what is at risk?

The dangers of not investing back into digital infrastructure can be divided into two categories: direct and indirect costs.

***Direct costs** include unspotted bugs and security vulnerabilities that could be exploited for malicious intent or lead to unexpected breaks in software functionality. These costs are acutely felt and cause problems that need to be immediately addressed.*

***Indirect costs** include things like loss of qualified labor and slower growth and innovation. While they are not immediately obvious, they represent uncaptured social value.*

Bugs, security vulnerabilities, and interruptions in service

The introduction to this report profiled the security bug Heartbleed, which was discovered in April 2014 in a software library called OpenSSL. Heartbleed, because it was so widespread and affected so many major websites, drew significant public attention to the security vulnerabilities in software.

In September 2014, another major security vulnerability was found in a key tool called Bash. Bash is included in popular operating systems like Linux and Mac OS, and as a result, is installed on more than 70% of the machines connected to the Internet.⁹⁶ The set of security bugs, dubbed “Shellshock,” could be exploited to allow someone unauthorized access to a computer system. The vulnerabilities had gone undetected for at least a decade. Bash was originally authored by a developer named Brian Fox in 1987, but since 1992 has been maintained by a single developer, Chet Ramey. He works as a senior technology architect at Case Western University in Ohio.

96

<http://timesofindia.indiatimes.com/tech/tech-news/Security-experts-expect-Shellshock-software-bug-to-be-significant/articleshow/43657819.cms>

Another project, OpenSSH, provides a free suite of security-related programs with widespread use across the web. Developers have discovered multiple security vulnerabilities in its code that have been subsequently addressed and fixed, including one in July 2015 that could allow attackers to bypass limits on password login attempts, and one in January 2016 that could leak private security keys.^{97 98}

97

<http://www.scmagazineuk.com/openssh-flaw-opens-the-door-to-brute-force-attackers/article/428304/>

98

<http://arstechnica.com/security/2016/01/bug-that-can-leak-crypto-keys-just-fixed-in-widely-used-openssh/>

Part of the problem is that many open source projects are legacy tools, built once by a passionate developer or group of developers, who then lacked resources to manage their project’s success. Over time, contributions decline as others get bored and move on, but the project is still in active use, leaving one or two people to figure out how to keep it alive.

Another growing issue in today’s software world, with so many new and inexperienced developers, is that security concepts are not taught or prioritized. New developers simply want to write code that works; they don’t know how to make software secure, or they mistakenly assume that the public code they use in their software has

been audited for security. Even best practices around safely disclosing or managing vulnerabilities are not commonly taught or understood. Security only becomes an issue once a developer's code has become compromised.

Christopher Allen coauthored the first version of the Transport Layer Security (TLS) protocol, whose subsequent versions became a security standard used almost universally online, including by websites like Google, Facebook and YouTube. Although today it is a standard, of its origins, Christopher writes:

*“As the co-author of TLS I would not have predicted 15 years later that over half of the Internet would be using an implementation of TLS maintained by a 1/4 time engineer. This lack of support led to the infamous Heartbleed bug. I tell my cryptocurrency colleagues this story today to warn them that their leading edge crypto today may be ‘boring’ in a decade and suffer the same fate as it will no longer be exciting and their future hard work may be compromised.”*⁹⁹

Finally, the stability of our software potentially relies upon the good faith and cooperation of hundreds of developers, which introduces significant risk. The fragility of our digital infrastructure was recently demonstrated by a developer named Azer Koçulu.

Azer, a Node.js developer, hosted a number of libraries on a package manager platform called npm. After a conflict with npm over trademark rights on one of his projects, Azer, frustrated with the outcome, decided to remove everything he had ever published to npm.¹⁰⁰ One of those libraries, left-pad, was referenced in hundreds of other software projects. Although it was just a few lines of code, by

99

<https://medium.com/@christophera/i-ve-been-working-to-address-this-gap-for-a-while-thus-my-recent-exploration-of-the-commons-in-my-8094d41a874a#qyh31ida4> Quote edited for clarity by source.

100

<https://medium.com/@azerbike/i-ve-just-liberated-my-modules-9045c06be67c#4sdbklvqv>

removing the left-pad project, Azer broke countless other software developers' processes. Azer's decision caused so many issues that npm made the unprecedented decision to republish his library, against Azer's will, in order to restore functionality to the rest of the ecosystem.¹⁰¹ Npm also revised its policies to make it harder for developers to remove their libraries without warning, recognizing how individual actions could negatively affect so many others.¹⁰²

101

http://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/

102

<http://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm>

Software not getting the necessary maintenance it needs

Building digital infrastructure in a haphazard fashion means that all software gets built more slowly and inefficiently. One example of this can be found in the history of Python infrastructure.

An important infrastructure project for Python developers is called Setuptools. Setuptools provides a set of tools that make writing Python easier and more standardized.

Setuptools was written by a developer named PJ Eby in 2004. Over the next four years, Setuptools saw widespread adoption. However, Setuptools was difficult to implement and use, and Eby was largely unreceptive to outside contributions and fixes, wanting to maintain—as its original author—final say over Setuptools. In 2008, a group of developers, led by Tarek Ziade, decided to fork the project to force Eby to make improvements. They called the new project Distribute.

In 2013, these projects were once again merged under Setuptools. The multi-year rift, however, underscored both the dubious state of Python's infrastructure tools, and the difficulty of making improvements—in part because there was nobody dedicated and willing to address the community's problems.

Python tools began to improve once the working group Python Packaging Authority (PyPA) was formed to focus specifically on setting better standards for packaging. One developer, Donald Stufft, made Python packaging tools his primary focus and was hired by HP (now under HPE) in May 2015 to continue his work. (His story will be discussed later in this report.)¹⁰³

103

Email interview with Russell
Keith-Magee and Hynek
Schlawack

Another example involves RubyGems.org, a website that most Ruby developers use to host their Ruby libraries. Ruby has been used to build major websites including Twitter, AirBnB, YellowPages, and GitHub itself.

In 2013, a security flaw in RubyGems.org was discovered, but went unfixed for several days, because RubyGems.org was maintained entirely by volunteers. The volunteers planned to address it that weekend, but in the meantime someone else discovered the flaw and hacked the RubyGems.org server. Following the hack, the servers had to be rebuilt from scratch. Several volunteers took time off work, and some even took personal vacation days, in order to get RubyGems.org up and running again as soon as possible. Because RubyGems.org is a critical piece of Ruby infrastructure, the security issue affected many developers and companies in turn.

The incident highlighted how pure volunteer labor limited the amount of security and reliability that could be provided to important software infrastructure. Dozens of developers “volunteered” during the incident, since the problem affected their regular jobs. Unfortunately, none of them had the previous experience needed to be helpful, and none of them continued to offer to help once the servers were repaired. In 2015, an organization called Ruby Together was formed to help pay pay for maintenance and development of

Ruby's infrastructure, including RubyGems.org, using company sponsorships.¹⁰⁴

104

Phone and email interview with André Arko. Thanks to André for his edits.

Loss of qualified labor

Like any community of volunteers, burnout is common among open source contributors, who find themselves fielding requests, from both individual users and companies, for work without compensation.

Many developers have stories about getting requests from companies for free work. Daniel Roy Greenfeld, a Python and Django developer, wrote:

*“I personally get regular demands for unpaid work (Discussions about payment for work always stall) by healthy high profit companies large and small for [my projects]. If I don't respond in a timely fashion, if I'm not willing to accept a crappy pull request, I/we get labeled a jerk. There is nothing like having core Python/PyPA maintainers working for Redhat [sic] demanding unpaid work while criticizing what they consider your project's shortcomings to ruin your day and diminish your belief in open source.”*¹⁰⁵

105

<https://github.com/pybee/paying-the-piper/issues/26>

(Red Hat is a multinational software company with annual revenue exceeding \$2B which sells open source software solutions to enterprise customers.¹⁰⁶ Because of the nature of their business, Red Hat employees use and contribute to many open source projects; in some ways, Red Hat has become the business poster child of open source. The company's financial success will be discussed later in this report.)

106

<http://fortune.com/2016/03/22/red-hat-revenue-2-billion-open-source/>

Read the Docs, the aforementioned documentation hosting service, explicitly states on its website that they do not support custom company installations and private support.¹⁰⁷ One of the maintainers, Eric Holscher, went as far as to comment, “*Pretty sure there is very little value in Read the Docs being open source, as private users never contribute back, only ask for free support.*”¹⁰⁸

107

<http://docs.readthedocs.io/en/latest/open-source-philosophy.html>

108

<https://twitter.com/ericholscher/status/689569190043201536>

Marquess, the OpenSSL contributor, made a tongue-in-cheek remark to these repeated requests in his post about funding:

*“I’m looking at you, Fortune 1000 companies. The ones who include OpenSSL in your firewall/appliance/cloud/financial/security products that you sell for profit, and/or who use it to secure your internal infrastructure and communications. The ones who don’t have to fund an in-house team of programmers to wrangle crypto code, and who then nag us for free consulting services when you can’t figure out how to use it. The ones who have never lifted a finger to contribute to the open source community that gave you this gift. You know who you are.”*¹⁰⁹

109

<http://veridicalsystems.com/blog/of-money-responsibility-and-pride/>

Some developers choose to stop maintaining their projects because they no longer have the time to dedicate to it, and hope that somebody else picks up where they left off. Meanwhile, companies, governments and individuals depend on these libraries for their continued use, unaware of the underlying situation.

David Michael Ross, an engineering manager at a web agency, wrote of his experience:

“That’s the big thing for me. [...] It’s knowing you did something for free, out of love, and there’s an endless stream out people

going ‘more! more!’ and getting angry when you won't accommodate their edge case.

I had my phone number on one of my personal sites so friends could get in touch with me. Took it down after a week because people would call me in the middle of the day for plugin support, even though there's a forum for support. There's nothing inherently wrong with that, it just wears you down. Makes you afraid to check email or answer the phone.¹¹⁰

110

<https://news.ycombinator.com/item?id=8712370>

Ryan Bigg, who writes documentation for the software framework Ruby on Rails, announced in November 2015 that he was quitting all open source work, explaining:

“I do not have the time or energy to invest in open source any more. I am not being paid at all to do any open source work, and so the work that I do there is time that I could be spending doing ‘life stuff,’ or writing. It is not fair to expect me to do even more work outside of my regular work, and then not get fairly compensated (time or money) for it. It's also a great recipe for burnout and making me just generally grumpy.¹¹¹”

111

<http://ryanbigg.com/2015/11/open-source-work/>

Loss of qualified labor also does not just refer to open source contributors who quit, but those who never join in the first place.

There are very few statistics on the demographics of open source contributors, which is telling in itself. A recent analysis of GitHub found that just 5.4% of open source contributors were women, compared to roughly 15 to 20% of technical roles at software companies overall.¹¹²

112

The 5.4% figure refers to open source contributors with more than ten contributions. <http://www.toptal.com/open-source/is-open-source-open-to-women>

One reason why open source contributors are strikingly more homogenous than the technology sector at large is that they need time and money to make significant contributions in the first place. These constraints prevent otherwise qualified contributors from entering the space.

David Maclver, creator of Hypothesis, a Python library for testing software applications, explains why he was able to spend so much time on the project:

*“I could only do this because I had the time and money to do so. I had the time to do so because I was being obsessive, had no dependents, and didn’t have a job. I could only not have a job because of the money. I only had the money because I spent the latter half of last year with double the salary I was used to, half the living expenses I was used to, and too borderline depressed to spend it on anything interesting. These are not reasonable requirements. [...] Can you produce quality software in less time than that, working only in your free time? I doubt it.”*¹¹³

113

<http://www.drmacliver.com/2015/04/its-ok-for-your-open-source-library-to-be-a-bit-shitty/>

Cory Benfield, a core Python developer, writes:

*“Generally speaking, people who aren’t cisgender, heterosexual, white, middle-class, English-speaking men are less able to tolerate the increased financial risk of not having a steady job. This means that those individuals really need a steady pay cheque to contribute most effectively. And we *need* those contributors: diverse teams make better things than homogeneous teams.”*¹¹⁴

114

Email interview with Cory Benfield

Charlotte Spencer, a contributor to software framework Hoodie and database PouchDB, echoes these sentiments:

*All my contributions are purely volunteered. I don't make any money, though I would absolutely like to. I have asked veteran open sourcerers if they are paid and they say they are not, which discouraged me from pursuing anything (if they aren't paid, why would I be?). I use most of my free time to do it, which I'm trying to do less of as it was taking up my life.*¹¹⁵

115

Email interview with Charlotte Spencer

Jessica Lord, a developer, actively contributed to open source while working at Code for America, a nonprofit organization that supports technology in the public sector. An urban planner by education, she stresses that she had “no computer science degree, no real production programming [sic] experience but a GitHub portfolio.”¹¹⁶ Her regular contributions drew the attention of GitHub itself, where she now works today.

116

<http://jlord.us/blog/osos-talk.html>

However, Jessica points out that she was able to contribute to open source under a “privileged” set of circumstances: she took a pay cut to work at Code for America, burned through her savings, worked “nearly constantly” on open source projects, and had a community of support.

Of lack of diversity in open source, Jessica writes:

The value of common knowledge cannot be overestimated. We must do better. We need all the ideas from all the people. That's what we should be aiming for.

*We need an open source for everyone. Not just for the privileged and not even just for the developers.*¹¹⁷

117

<http://jlord.us/blog/osos-talk.html>

Jessica's last point also underlines how enabling more diverse perspectives in open source can help sustain open source itself. From a functional perspective, the vast majority of open source contributors are developers, but plenty of other roles are needed to sustain larger projects, including writing, project management, and outreach. Open source projects are not dissimilar from other types of organizations, including startups, where administration, marketing, design, and other roles are needed to support an organization's raw output. It is partially because open source culture is so heavily weighted to developers that sustainability is rarely discussed or acted upon.

Finally, the homogeneity of open source contributors impacts diversity efforts in technology at large, because it is so closely tied to hiring. As previously mentioned, many employers use open source contributions, including GitHub profiles, to discover potential new hires or to check a candidate's qualifications. Employers who rely heavily upon evidence of open source contributions are drawing from an extremely narrow pool of candidates.

Ashe Dryden, in an influential essay called "The Ethics of Unpaid Labor and the OSS Community," explained:


“Deciding that someone is a good programmer based solely on their publicly available code excludes far more than marginalized people. It also excludes anyone who can't release their code publicly because of licensing or security reasons. This also includes a large number of freelancers and contractors who are unable to publicly claim that they worked on a project for legal

*reasons (NDAs, for instance). In an industry where we are struggling to find enough talent, why are we artificially limiting the talent pool?*¹¹⁸


118

<http://www.ashedryden.com/blog/the-ethics-of-unpaid-labor-and-the-oss-community>

How can we mitigate or avoid some of these costs associated with participating in the building of digital infrastructure today? First, let's look at how infrastructure projects are currently supported.



Sustaining Digital Infrastructure



"While OpenSSL does 'belong to the people' it is neither realistic nor appropriate to expect that a few hundred, or even a few thousand, individuals provide all the financial support."

– Steve Marquess, OpenSSL

Business models for digital infrastructure

Some parts of digital infrastructure function well in a business context. Databases and hosting services, for example, tend to be well-funded and profitable businesses, because they can charge for access. Like electricity or water utilities, access to a server or database can be metered, charged for, and shut off if fees go unpaid.

Heroku (mentioned at the beginning of this paper) and Amazon Web Services are two prominent examples of platforms that sell digital infrastructure services to software developers for a fee. (Note that neither project is open source.) Similar open source projects at this level of infrastructure, such as OpenStack (a platform competitive to Amazon Web Services) or MySQL (a database), have found corporate homes. OpenStack is funded by a consortium of companies and MySQL was acquired by Oracle.

Part of what makes these services financially attractive is the lack of noise. A developer may use 20 different libraries, all with different functions, in a single software application, but they only need one database. Therefore, successful projects are more likely to get the attention and care that they need.

Another helpful way of thinking about infrastructure that can be charged for is that if there is an immediate risk of downtime, it probably has a business model. In other words, a server can have unexpected interruptions in service, the way electricity might unexpectedly shut off, but a programming language does not “break” or have downtime in the same way, because it is a system of information.¹¹⁹

119

Thanks to Sam Gerstenzang for framing this distinction: <https://twitter.com/gerstenzang/status/687404438005366784>

For these types of open source projects, business models tend to revolve around finding services or support to charge for. This works for projects with significant enterprise use, particularly when the problem is technically complex, or a company needs a feature to be built.

Bounties

On a small scale, individuals or companies sometimes post “bounties”, which are cash prizes for certain development milestones. For example, IBM regularly requests new features for various projects using a website called Bountysource, offering up to \$5,000 per task. Bountysource is a popular platform to find and post bounties; it has over 26,000 members.¹²⁰

120

<https://www.bountysource.com/>

Bounties help address the aforementioned issues of simply donating to a project. Because bounties are clearly tied to an outcome, the money will get used. On the other hand, bounties can create perverse incentives for contributing to a project.

Bounties can dictate which work does or doesn't get done, and sometimes that work doesn't align with a project's priorities. It can also introduce noise into the system: for example, a company could offer an expensive bounty for a feature that the project owners do not consider important.

On the contributor side, outsiders with no knowledge of a project might jump in just to complete the bounty, then leave. Or, they might do a poor job of completing the request, because they are trying to collect bounties. Finally, bounties can be appropriate for funding new features or prominent bugs, but are less practical for

funding ongoing operations, such as customer support or maintenance.

Jeff Atwood, the creator of Stack Overflow, noted these problems with bounty programs, particularly with regard to security:

“*One unfortunate side effect of this bug bounty trend is that it attracts not just bona fide programmers interested in security, but anyone interested in easy money. We've gotten too many 'serious' security bug reports that were extremely low value. And we have to follow up on these, because they are 'serious,' right? Unfortunately, many of them are a waste of time...The incentives feel really wrong to me. As much as I know security is incredibly important, I view these interactions with an increasing sense of dread because they generate work for me and the returns are low.*¹²¹

121

<http://blog.codinghorror.com/given-enough-money-all-bugs-are-shallow/>

Services

On a larger scale, one of the best-known and oft-cited examples of an open source business model is Red Hat, the aforementioned enterprise company, which offers support, training and other services to enterprises that use Linux. Red Hat was founded in 1993 and is a publicly traded company with reported revenues of \$2B per year.

Although Red Hat has been wildly successful from a financial standpoint, many are quick to point out that it is an anomaly unlikely to be repeated again. Red Hat benefited from first mover advantage for the technology it supports. Matt Asay, a journalist who focuses on open source, noted that Red Hat uses a unique set of patents and

licensing to protect its enterprise market. Asay, once a major proponent of open source businesses, now believes that some proprietary licensing is necessary to build a serious business.¹²² Matthew Aslet of the 451 Group, a research group, similarly found that most successful “open source” enterprises actually use some form of commercial licensing.¹²³

122

http://www.theregister.co.uk/2011/03/29/red_hat_billions/

123

<https://blogs.the451group.com/open-source/2008/10/13/open-source-is-not-a-business-model/>

Docker, the previously mentioned open source project that helps applications run efficiently, is a more recent example of a company attempting this model. Docker has raised \$180M in venture capital from investors, with a reported \$1B valuation from private investors.¹²⁴ As their market share has grown, Docker has begun offering enterprise-level support services. Yet without strong revenues, Docker could simply be another example of venture capital making a “loss leader” infrastructure investment.

124

<https://www.crunchbase.com/organization/docker#/entity>

On a smaller scale, many developers offer consulting services to subsidize their work. Hoodie is a lightweight software framework based on Node that has found success with consulting services. Hoodie itself is an open source project. Several maintainers earn money through a boutique firm, Neighbourhoodie, that offers software development services.¹²⁵ Although Neighbourhoodie specializes in the Hoodie framework, Hoodie is still a fairly new project, so some of their work comes from non-Hoodie-related projects.¹²⁶ In Hoodie’s case, the services model is meant to support the salaries of several maintainers, rather than aiming for a Red Hat-sized enterprise strategy.

125

<http://neighbourhoodie.ie/>

126

Phone call interview with Jan Lehnardt

Consulting is a viable option for independent developers, if there are enough people using the project who are willing and able to pay for extra help. However, on a small scale, it can also distract developers

from improving the project itself, as the one or two maintainers are now spending their time building a business and providing services that may or may not align with a project's maintenance needs.

Aiming for a consulting business can also put creators at odds with making the product easy to use and get started with, which is in the spirit of open source. Twisted, the aforementioned Python library, shared a playful testimony from one of its users, a company called Mailman: *"You guys have a big problem, because it was way too easy to do! How are you going to make the big consulting bucks? :)"*¹²⁷

127

<https://twistedmatrix.com/trac/wiki/SuccessStories>

In the end, the "business model" for an open source project is not dissimilar from simply freelancing.

Paid licenses

Some developers feel that licensing could provide at least a partial solution to open source's funding problems. If open source projects are being heavily used, why not charge for them?

These "paid licenses" are not technically open source licenses, according to the Open Source Initiative's definition.¹²⁸ Rather, they are initiatives attempting to balance the very real need for paid work with the desire to make code available to the public. This type of code might be called "source visible" or "source available." Fair Source, for example, describes itself as *"[offering] some of the benefits of open source while preserving the ability to charge for the software."*¹²⁹

128

<https://opensource.org/definition>

129

<https://fair.io/>

The Fair Source license was announced in November 2015 by a company called Sourcegraph to address the need for a paid license.

The license terms were drafted by Heather Meeker, a lawyer who also worked on the core team for the Mozilla Public License v2.0.

Under Fair Source, code is free to view, download, execute, and modify up to a certain number of users in an organization. After that limit is reached, the organization must pay a licensing fee, determined by the publisher.¹³⁰ In other words, Fair Source code is free for personal and small business use, but provides a legal basis to charge for bigger commercial use cases.

130
<https://fair.io/>

Sourcegraph's announcement of the Fair Source license, which they themselves now use, sparked a spirited debate about monetizing open source. (It is worth noting that a comparable "shareware" movement was attempted and popularized in the 1980s.)

Mike Perham, a maintainer of Sidekiq, a popular Ruby developer tool, also recently suggested that open source contributors use a "dual license" to monetize their work, charging companies for access to a permissive MIT license instead of a more restrictive AGPL license that would require attribution. His theory is that by making AGPL the default license, "businesses will pay to avoid it."

To support this idea, Perham reminded his audience:

Remember: Open Source != Free Software. The source may be viewable on GitHub but that doesn't mean anyone can use it for any purpose.¹³¹ There's no reason you can't make your source code accessible but also charge to use it. As long as you are the owner of the code, you have the right to license it however you want.

...[The] reality is most smaller OSS projects have a single person

131
 It is worth noting that Mike is correct about code hosted on GitHub with no specified license, but an open source license that meets the OSI's definition must include freedom to redistribute. This quote highlights how the modern definition of "open source" is blurring, with colloquial use becoming distinct from historical definition.

*doing 95% of the work. If this is true, be grateful for unpaid help but don't feel guilty about keeping 100% of the income.*¹³²

132

<http://www.mikeperham.com/2015/11/23/how-to-charge-for-your-open-source/>

Charging businesses provides another option for developers to support their work, particularly if it is one or two maintainers supporting an active project. However, not every project could successfully charge for its work, especially older projects, or infrastructure projects that look more like public goods than consumer products, such as programming languages.

While paid licensing could work for certain product scenarios, this model is also arguably at odds with the enormous social value that open source has provided, which suggests that when software is free, innovation follows. The goal should not be to move back towards a closed software society, where progress and creativity are constrained, but to sustainably support a public ecosystem in which software can be freely created and distributed.

Finding a sponsor or donor for an infrastructure project

The other option for supporting infrastructure projects is to find sponsorships or donations. This practice is especially common in the following situations:

There is no paid client demand for services associated with the project

Charging directly for access would prevent adoption (one could not charge to use Python as a programming language, for example, because nobody would use it; it would be like charging people to speak English)

There isn't enough capacity to manage paid work, or no desire on the developer's part to deal with business matters

Neutrality and non-commercialization are perceived to be important for governance purposes

In this situation, a project maintainer will look for benefactors who believe in the value of their work and are willing to financially support them. There are two major sources of funding at the moment: software companies and other developers.

Crowdfunding

Some development work gets funded through crowdfunding campaigns, such as Kickstarter or Indiegogo. Bountysource, the aforementioned open source bounty website, also has a platform called

Salt, dedicated to crowdfunding open source projects.

Andrew Godwin, a London-based Django core developer, successfully raised £17,952 (roughly \$25,000) from 507 backers on Kickstarter to fund database work for Django. The project was fully funded in less than four hours.

Of his decision to raise funds for an open source project, Godwin wrote:

“A lot of open source code gets done for free. However, my free time is limited. I currently have a day a week available for work, and I'd love to spend it improving Django rather than doing consulting or contracting.

*The idea here is twofold—to guarantee the project a solid period of work and at least 80 or so hours of coding time, as well as to try and show the world that open source software really can pay for developers' time.*¹³³

133

<https://www.kickstarter.com/projects/andrewgodwin/schema-migrations-for-django/description>

Similarly to bounties, crowdfunding can be useful for funding new features or development work with a clear, tangible outcome. Crowdfunding can also help reduce perverse incentives from bounties, since campaigns take a bit more time and effort than posting a bounty, and tend to get funded based on public trust in the campaign owner's ability to deliver. In this case, Godwin had been a core contributor to Django for six years and was well-known in the community.

However, crowdfunding still does not address the need for funding related to ongoing operations and overhead. It is also not a

recurring source of capital, and there is a mental and time cost associated with planning and executing a crowdfunding campaign each time. Finally, backers of these projects tend to be fellow developers, or small company contributions—there are only so many times a campaign owner can tap the same source of capital to fund their projects.

Godwin himself later wrote of the experience:

“I don't think [crowdfunding] meshes entirely well with general open source development; not only is it just a one-time payment, but the idea of rewards often doesn't match well and it requires something you can entirely bound and describe up front.

Just relying on people's good will [sic] isn't going to work, and we'll end up disproportionately appealing to independent developers or developers on a personal level and that's not as sustainable I don't think.”¹³⁴

134

<https://github.com/pybee/paying-the-piper/issues/3>

Aside from crowdfunding campaigns, several platforms have also emerged to encourage the practice of “tipping” open source contributors: that is, pledging small amounts of recurring revenue to a contributor as a sign of support for his or her work. Two popular platforms are Patreon (which does not exclusively focus on open source contributors) and Gratipay (which tends to attract a more technical community).

The idea of recurring revenue is appealing, but suffers some of the same problems as crowdfunding. Namely, most patrons are developers themselves, with limited amounts of capital to pledge to each other. Donations are widely considered to be helpful as “beer money,”

but not “rent money.” Gratipay has 122 teams on its platform who collectively receive \$1,000 per week, meaning that the average project receives less than \$40 per month.¹³⁵

135

As of November 2015. <https://gratipay.com/>

Even very large projects such as OpenSSL only generated \$2,000 in annual donations prior to Heartbleed. As mentioned before, after Heartbleed, team member Steve Marquess noted the “outpouring of grassroots support from the OpenSSL community”: the first round of donations came out to roughly 200 donors for a total of \$9,000. Marquess thanked the community for its support, but also noted:

“Even if those donations continue to arrive at the same rate indefinitely (they won’t), and even though every penny of those funds goes directly to OpenSSL team members, it is nowhere near enough to properly sustain the manpower levels needed to support such a complex and critical software product. While OpenSSL does ‘belong to the people’ it is neither realistic nor appropriate to expect that a few hundred, or even a few thousand, individuals provide all the financial support. The ones who should be contributing real resources are the commercial companies and governments who use OpenSSL extensively and take it for granted.”¹³⁶

136

<http://veridicalsystems.com/blog/of-money-responsibility-and-pride/>

(To Marquess’s point, subsequent corporate donations were larger, as companies had more to give than individuals. The biggest donation came from Chinese smartphone maker Smartisan, to the tune of \$160,000.¹³⁷ Smartisan has continued to make substantial donations to OpenSSL.¹³⁸)

137

<http://blogs.wsj.com/cio/2014/08/20/openssl-seeing-more-support-post-heartbleed/>

138

Email interview with Steve Marquess

Finally, the reality is that there are too many projects, all valuable or critical in some way, and not enough donors, for the technical

community—companies or individuals—to be able to lend mindshare and donate significantly to all of them.

Corporate sponsorship of infrastructure projects

On a larger scale, in some instances, a project's value becomes so widely regarded that a company will hire a contributor to work on the project full-time.

John Resig is the author of jQuery, a JavaScript programming library that is used by nearly two-thirds of the top million most-trafficked websites.¹³⁹ He built and released jQuery as a side project in 2006. John joined Mozilla in 2007 as a developer evangelist, specifically focusing on JavaScript libraries.¹⁴⁰

139

<http://libscore.com/#libs>

140

<http://ejohn.org/blog/mozilla/>

As jQuery continued to grow in popularity, it became apparent that, in addition to further technical development, it needed to formalize some of its governance aspects. Mozilla suggested that John work on jQuery full-time from 2009 to 2011, which he did.

Of the experience, John wrote:

*“Over the past year and a half Mozilla gave me the ability to work on jQuery full-time. This has resulted in 9 releases of jQuery...and a drastically improved jQuery organization (we're now under the Software Freedom Conservancy non-profit, hold frequent team meetings, public votes, provide public status updates, and actively encourage participation). Thankfully the jQuery project is running quite smoothly these days, allowing me to scale back my involvement to a more-reasonable amount of time and take on other development work.”*¹⁴¹

141

<http://ejohn.org/blog/next-steps-in-2011/>

After using his time at Mozilla to get jQuery the organizational support it needed, John announced he would join Khan Academy to focus on new projects outside of jQuery.

Cory Benfield, a Python developer, has a similar story. Cory contributed to open source projects in his spare time, eventually becoming a core developer for a critical Python library called Requests.

Benfield notes:

“This library is up there with Django in terms of being ‘critical infrastructure’ for Python developers, and yet before I came on to [sic] the project was essentially maintained by a single individual.”¹⁴²

142

Email interview with Cory Benfield

Benfield estimates that he volunteered on the project roughly 12 hours per week for nearly four years, in addition to his full-time job.¹⁴³ Nobody was paid to work on Requests.

143

Email interview with Cory Benfield

During this time, HP hired an employee, Donald Stufft, to focus specifically on supporting Python-related projects, since HP considers Python to be critical to its software. (Donald is the aforementioned developer who is paid full-time to focus on Python packaging.) Donald convinced his manager to hire Cory to focus full-time on Python projects, where he currently works today (now under HPE).

Companies are well-positioned to financially support volunteer projects that they consider to be critical to their business, and when situations like John Resig’s or Cory Benfield’s happen, they are warmly received. However, there are complications.

Firstly, no company is obligated to hire someone to work on projects in these situations; they tend to come about by chance from a sympathetic patron. Once an employee is hired, there is always the possibility of losing that sponsorship, especially because the employee does not directly contribute to the company's bottom line. This is especially dangerous if a project's sustainability depends upon one contributor being employed full-time. In the case of Requests, Cory is the only full-time contributor (there are two other part-time contributors, Ian Cordasco and Kenneth Reitz).

One situation where this occurred was in the case of rvm, a critical piece of Ruby infrastructure. Michal Papis, its primary author, was hired by Engine Yard to support rvm's development from 2011 to 2013. When that sponsorship ended, however, Papis had to run a crowdfunding campaign to support rvm's ongoing development.¹⁴⁴

144

<https://www.bountysource.com/teams/rvm/fundraiser>

It wasn't just rvm, either. Engine Yard had employed a number of maintainers of Ruby infrastructure projects, including JRuby, Ruby on Rails 3, and bundler. When Engine Yard was forced to make the right business decision for their company, which was to scale back support, all of these projects lost full-time maintainers, nearly all at the same time.

Another concern is that a single company could have undue influence over the project, since they are effectively the only sponsor. Cory Benfield notes that that the contributor him or herself could also have undue influence over the project, since they now have much more time than others to make contributions.¹⁴⁵ In theory, that decision could be made between a company and a maintainer, without involving the project's larger community.

145

Email interview with Cory Benfield

One example where this occurred was in the case of Express.js, a critical framework for the Node.js ecosystem. When the original author decided to move on from the project, he transferred the assets (including the code repository and domain name) to a company called StrongLoop, whose employees agreed to help maintain the project.¹⁴⁶ However, StrongLoop didn't provide the support that the community expected, and because StrongLoop alone had administrative access, it became difficult for the community to contribute. Doug Wilson, a lead maintainer on the project who is unaffiliated with StrongLoop, still had commit access and continued to manage the project's workload, struggling to handle the project's needs by himself.

146

<https://strongloop.com/strongblog/tj-holowaychuk-sponsorship-of-express/>

Following StrongLoop's acquisition by IBM, Doug declared that StrongLoop had effectively killed the contributor community:

“At the time of the StrongLoop move, we had active members like @Fishrock123 working to create...documentation. Then all there was was me as a single person doing this in my free time with mounting support requests....all this time I've been killing myself, I have been committing under the StrongLoop name. No matter what happens, I will not ever commit again to any repository under StrongLoop's name.”¹⁴⁷

147

<https://github.com/strongloop/express/issues/2844#issuecomment-172414097>

Ultimately, the Express.js project was moved out of StrongLoop's administration and into the Node.js Foundation, which helps steward projects that are part of the Node.js technology ecosystem.

For very large and well-known open source projects, however, hiring developers is not an unusual practice. The Linux Foundation reported, for example, that more than 80 percent of Linux kernel

development is done by developers who are paid for their work.¹⁴⁸ The Linux Foundation also hires paid Fellows to work on infrastructure projects full time, including Greg Kroah-Hartman, a Linux kernel developer, and Linus Torvalds himself, the creator of the Linux kernel.

148

<http://www.linuxfoundation.org/news-media/announcements/2015/02/linux-foundation-releases-linux-development-report>

Why is it so hard to fund these projects?

Today, infrastructure work is being cobbled together by freelance developers or those with “day jobs,” doing unrelated paid client work for part of the month and working on open source projects in their spare time. While this is a viable way to pay for one’s lifestyle, it does not adequately reflect the social value that these projects deserve.

Stunningly, although everybody agrees there is a problem (whether defined as “volunteer burnout”, community mismanagement or a greater lack of funding), the conversation has not progressed beyond meager, short-term solutions such as tipping or crowdfunding.

Talk to developers who found a way to pay themselves, and you’ll hear the word “lucky” thrown around: lucky to have been hired by a company, lucky to have gotten publicity and donations, lucky to have stumbled upon a business model, lucky to not have a family or mortgage to worry about. Everybody is getting lucky. Luck lasts for a couple of months, maybe a year or two, and then it runs out.

Why is it so hard to fund digital infrastructure?

Fundamentally, digital infrastructure has a free rider problem. Resources are offered for free, and everybody (whether individual developer or large software company) uses them, so nobody is incentivized to contribute back, figuring that somebody else will step in. Left unchecked, this will lead to a tragedy of the commons.

In addition to the macroeconomic challenge of the commons, there are several reasons why supporting digital infrastructure is particularly complicated. Although they have been touched upon throughout this report, they are summarized here:

There is a misperception that this is a “solved problem.” The pervasive belief, even among stakeholders such as software companies, that open source is well-funded, makes it harder to generate support. Some infrastructure projects operate sustainably, either because they have a working business model or sponsorship, or because their required upkeep is limited. An unfamiliar audience will also associate open source with enterprise companies like Red Hat or Docker and assume the problem has been solved. However, these situations are the outliers, not the rule.

There is a lack of cultural understanding and awareness about the problem. Outside of the open source community, nearly everybody remains unaware of infrastructure’s funding issues, and the topic is perceived to be dry and technical. Developers needing support tend to have a highly technical focus and aren’t comfortable advocating for the business side of their work. Taken together, there is no momentum to change a broken situation.

Digital infrastructure is rooted in open source, whose volunteer culture discourages talk of money. Although this attitude has made open source what it is today, it also makes it difficult for developers to openly discuss their needs without feeling guilty or worrying about not being perceived as a team player. Open source’s highly distributed and democratic nature also makes it difficult to coordinate and sustain institutional actors who could act as advocates for their needs.

Digital infrastructure is highly distributed, compared to physical infrastructure. Unlike planning the construction of a bridge, it’s not always clear which projects are useful until after they have already taken off.

They cannot be planned beforehand by a centralized entity. At the other end of the lifecycle, some projects are meant to decline as other, better solutions take their place. Digital infrastructure is distributed across hundreds of projects, large and small, built by individuals, groups and companies; it would be a behemoth task to catalog them all.

“It's hard to find funding...for the average developer (me) some of them are totally out of reach. [Kickstarter] only works if you either go viral or hire someone to do all of the marketing/design/promotions....Turning a project into a business is great too, but...these are all things that take away from development (which is the part I like to focus on). If I wanted to get a grant, I wouldn't even know where to start.”¹⁴⁹

149

<https://github.com/pybee/paying-the-piper/issues/53>

- Kyle Kemp, freelance developer and open source contributor

Institutional efforts to support digital infrastructure

There are some institutional efforts to collectively organize and help support open source projects. Some are independent software foundations; other sources of support come from software companies themselves.

Administrative support and fiscal sponsorship

Several foundations provide organizational support, such as fiscal sponsorship, to open source projects: in other words, taking care of the non-coding tasks that many developers would prefer not to do.

The Apache Software Foundation, incorporated in 1999, was created partially to support development of the Apache HTTP Server, which serves roughly 55% of all websites worldwide.¹⁵⁰ Since then, it has become a home for over 350 open source projects.¹⁵¹ Apache structures itself as a decentralized community of developers, with no full-time employees and nearly 3,000 volunteers. It offers several services to projects, mostly around organizational, legal, and development support. As of 2011, Apache had an annual budget of over \$500,000, most of which comes from grants and contributions.¹⁵²

The Software Freedom Conservancy, established in 2006, also provides nonprofit administration services for over 30 free and open source projects. Projects supported by the Software Freedom Conservancy include Git, the aforementioned version control system upon which GitHub built its platform, and Twisted, the aforementioned Python library.¹⁵³

150

http://w3techs.com/technologies/overview/web_server/all

151

<http://www.apache.org/>

152

https://en.wikipedia.org/wiki/Apache_Software_Foundation

153

<https://sfconservancy.org/members/current/>

Other examples of foundations who provide organizational support include The Eclipse Foundation and Software in the Public Interest. The Linux Foundation and Mozilla Foundation also support external open source projects in various ways (discussed later in this section), though that is not the primary purpose of their mission.

It is important to note that these foundations provide legal and administrative, but rarely financial, support. Therefore, sponsorship by Apache or the Software Freedom Conservancy alone does not fund a project in itself; the foundations only help make it easier to process donations and manage the project.

Another important observation is that these initiatives support free and open source software from a philosophical perspective, but do not focus on infrastructure specifically. For example, OpenTripPlanner, a project supported by the Software Freedom Conservancy, is trip planner software; although the code that powers it is open source, it is a consumer application, not infrastructure.

Creating a foundation to support a project

Some projects are large enough to be managed through their own foundations. Python, Node.js, Django, and jQuery all have complementary foundations.

There are two important aspects to getting a foundation off the ground: qualifying for tax-exempt status and finding funding.

Qualifying as a 501(c)(3) can be challenging for these projects, due to the lack of awareness about open source technology and

tendency to see open source as a non-charitable activity. In 2013, a controversy revealed that the IRS had internally identified a list of groups applying for tax-exempt status that would require further scrutiny; “open source” was one of these.^{154 155} Unfortunately, these constraints make it difficult for projects to institutionalize.

154

https://en.wikipedia.org/wiki/IRS_targeting_controversy

For instance, Russell Keith-Magee, who until recently was president of the Django Software Foundation, explained that the foundation cannot directly “fund” software development of Django, without the risk of losing its 501(c)(3) status. Instead, they “support” its development through community activities.

155

<http://www.forbes.com/sites/kellyphillips/2014/07/17/not-just-the-tea-party-irs-targeted-turned-down-tax-exempt-status-tied-to-open-source-software/>

In June 2014, the Yorba Foundation, which made Linux productivity software, was denied 501(c)(3) status, after waiting nearly four and a half years for the decision. Jim Nelson, its executive director, was particularly alarmed by the IRS’s reasoning: because their software could be used by commercial entities, Yorba’s work could not be considered charitable. A letter from the IRS explained:

*“Mere publishing under open source licenses for all to use does not show that the poor and underprivileged actually use the Tools.[...]You do not know who uses the Tools much less what kind of content they create with the Tools.”*¹⁵⁶

156

<https://blogs.gnome.org/jnelson/2014/06/30/the-new-501c3-and-the-future-of-free-software-in-the-united-states/>

Nelson pointed out the flaws in this reasoning in a blog post, comparing it to any other public good:

*“There’s a charitable organization here in San Francisco that plants trees throughout the city for the benefit of all. If one of their trees...cools the cafe’s patrons as they enjoy their espressos, does that mean the tree-planting organization is no longer a charity?”*¹⁵⁷

157

<https://blogs.gnome.org/jnelson/2014/06/30/the-new-501c3-and-the-future-of-free-software-in-the-united-states/>

Projects that qualify for 501(c)(3) status tend to mention their focus on community, as with the Python Software Foundation, whose mission is “*to promote, protect, and advance the Python programming language, and to support and facilitate the growth of a diverse and international community of Python programmers.*”¹⁵⁸

158

<https://www.python.org/psf/>

Alternatively, some projects apply to become a trade association through a 501(c)(6) status. The jQuery Foundation is an example of this, describing itself as a “*member supported non-profit trade association for web developers.*”¹⁵⁹ The Linux Foundation is also a trade association.

159

<https://jquery.org/>

The second aspect of formalizing project governance through a foundation is finding the right source of funding. Some foundations are supported by individual contributions, but have comparatively small budgets.

The Django Software Foundation, for example, manages Django, the most popular web framework written in Python, used by companies like Instagram and Pinterest. The foundation is run by volunteers and takes in less than \$60,000 in donations per year.¹⁶⁰ Last year, they received a one-time \$150,000 grant from the Mozilla Foundation.¹⁶¹

160

<https://www.djangoproject.com/foundation/reports/2013/>

161

<https://www.djangoproject.com/weblog/2015/dec/11/django-awarded-moss-grant/>

The other common source of funding is corporate sponsors. Corporate entities can be well-suited for funding because they use these software projects themselves. The Linux Foundation is one of the most successful outliers, due to the fundamental value of the Linux kernel to nearly every corporate entity. The Linux Foundation has \$30M in annually managed capital from corporate members such as IBM, Intel, Oracle, and Samsung, and is growing.¹⁶²

162

http://collabprojects.linux-foundation.org/sites/collab-projects/files/lf_collaborative_projects_brochure.pdf

Creating a foundation to support a project is sensible for very large infrastructure projects. It is less suitable for smaller projects, due to the work, resources, and ongoing corporate sponsorship needed to create a sustainable organization.

Node.js is a recent successful example of using a foundation to support a large project. Node.js is a JavaScript framework, developed in 2009 by Ryan Dahl and several other developers working at Joyent, a private software company. It became extremely popular, but began to suffer governance constraints due to Joyent's patronage, whom some felt could not fully represent an enthusiastic and fast-growing Node.js community.

In 2014, a group of Node.js contributors threatened to fork the project. Joyent tried to address governance issues by forming an Advisory Board for the project, but the project was forked anyway, under the name io.js.¹⁶³ In February 2015, an intent to form a 501(c)(6) organization was announced which would remove Node.js from Joyent's stewardship. The Node.js and io.js communities voted to work together under this new entity, called the Node.js Foundation. The Node.js Foundation, structured under the advisorship of the Linux Foundation, has a number of corporate sponsors who financially contribute to its budget, including IBM, Microsoft, and PayPal.¹⁶⁴ These sponsors see influential value in supporting the development of a popular software project that powers the web, and they have the resources to spare.

Another promising example is Ruby Together, an organization launched by several Ruby developers to support Ruby infrastructure projects. Ruby Together is structured as a trade association, in which corporate and individual donors pledge money to fund full-time

163

<http://www.infoworld.com/article/2835159/node-js/node-js-governance-model-pushed-as-forking-talk-en-sues.html>

164

<https://www.joyent.com/blog/introducing-the-node-js-foundation>

developers to improve core Ruby infrastructure. Donors elect a volunteer board of directors, who help decide which projects Ruby Together should work on each month.

Ruby Together was conceived by, and funds the work of, two developers: André Arko and David Radcliffe. As of April 2016, they now fund the work of four other infrastructure maintainers. Their monthly budget as of March 2016 was just over \$18,000/month, funded entirely by donations. Ruby Together was announced in March 2015 and is still a new project, but could serve as a blueprint for a more community-oriented model to fund work on other software infrastructure projects.¹⁶⁵

¹⁶⁵

<https://rubytogether.org/>

Corporate programs

Software companies support infrastructure projects in various ways. As beneficiaries of infrastructure projects, they contribute back by reporting bugs, suggesting or submitting new features, and providing other forms of feedback. Some companies encourage their employees to contribute to critical projects on company time. Many employees have significant contributor roles for external open source projects. For some employees, open source work is an explicit part of their job. Dedicating salaried time is one of the most important ways that companies contribute to open source.

Large companies like Google or Facebook are eagerly embracing open source as way to build trust and influence, because they are the only institutional actors large enough to absorb its costs without needing a financial return on investment. Open source projects help strengthen a company's influence, whether by releasing their own

open source projects or hiring key developers to work on an open source project full-time.

These practices are not limited to pure software companies, either. Walmart, for example, is a major supporter of open source, investing over \$2M in an open source project called hapi.¹⁶⁶ Eran Hammer, a senior developer at Walmart Labs, was quick to explain that “*open source ain’t charity*” and that “*the size of the company using [hapi] translated to ‘free’ engineering resources.*”¹⁶⁷ Dion Almaer, the former VP of engineering at Walmart Labs, noted that their commitment to open source helped with recruiting, building a strong company culture, and “a slew of areas of leverage.”¹⁶⁸

In terms of direct support for maintainers, sometimes companies hire a maintainer to work on full-time on an open source project. Companies also occasionally donate to project crowdfunding campaigns. For example, a recent Kickstarter campaign to fund core work on Django raised £32,650 (roughly \$50,000); Tom Christie, the campaign organizer, reported that 80% of total financing came from businesses.¹⁶⁹ However, these efforts are still ad hoc, and digital infrastructure as a CSR (corporate social responsibility) issue is not yet common among for-profit software companies. There is room for advocacy here.

One of the best-known corporate programs is Google’s Summer of Code (GSoC), mentioned earlier in this paper, which provides stipends to university students to work on open source projects for a summer. Students are paired with mentors to help familiarize them with the project. Summer of Code is maintained by the Open Source Programs Office at Google, and has funded thousands of students.¹⁷⁰

¹⁷¹ GSoC’s goal is to give students an opportunity to write code for

166

<http://www.infoworld.com/article/2608897/open-source-software/walmart-s-investment-in-open-source-isn-t-cheap.html>

167

<http://hueniverse.com/2014/08/15/open-source-aint-charity/>

168

<http://todogroup.org/blog/why-we-run-an-open-source-program-walmart-labs/>

169

<https://github.com/pybee/paying-the-piper/issues/50>

170

<https://developers.google.com/open-source/gsoc/>

171

https://en.wikipedia.org/wiki/Google_Summer_of_Code

open source projects; it is not focused on funding those projects themselves.

Last year, Stripe, a payments processing company, announced an “Open-Source Retreat,” offering stipends of up to \$7,500 per month for a three month retreat at Stripe’s offices.¹⁷² They initially intended to offer just two grants, but after receiving 120 applications, they expanded the program to four grantees.

172

<https://stripe.com/blog/stripe-open-source-retreat>

The grantees were enthusiastic about the experience. One of the grantees, Andrey Petrov, continues to maintain the aforementioned Python library called `urllib3`, now used by every Python developer. Of the experience, Andrey wrote,

“Publishing and contributing to open source is going to continue happening regardless whether I’m getting paid for it or not, but it will be slow and unfocused. Which is fine, it’s how open source work has always worked. But it doesn’t need to be this way. [...]”

If you’re a tech company, please allocate a budget for open source grants and sponsorships. Distribute it on Gittip¹⁷³ if you wish, or do what Stripe did and fund aggressive sprints towards some high-impact milestones.

173

Gittip is now called Gratipay. The product has been modified somewhat since the original publication of Andrey’s post.

*Consider this a formal call for sponsorship: **Please help fund urllib3 development.**¹⁷⁴*

174

<https://stripe.com/blog/stripe-open-source-retreat>

Stripe’s Open-Source Retreat can serve as a model for what a CSR program could look like. Stripe decided to offer the program for a second year in 2015. Despite the popularity of their program and its

warm reception by developers, this practice is still not common among other companies.

Corporate interest in open source is growing rapidly, and nobody can predict exactly what it might mean in the long term. Companies could address the long term support gap by directing human resources and money towards open source projects. There could be formal fellowship programs to connect companies with open source maintainers needing full-time support. Whereas a project's contributors used to be a mix of developers from different places, perhaps they will now be filled by a group of employees from a single company. Digital infrastructure might come to exist in a series of "walled gardens," each well-supported and technically open, but effectively championed by one company and its employees, by virtue of that company's limitless resources.

Taken to its extreme, however, this scenario doesn't bode well for innovation. Jeff Lindsay, a software architect who helped build the platform team for Twilio, a highly successful cloud communications company, mused in a podcast last year:

“Twilio is incentivized to make Twilio better, Amazon is incentivized to make Amazon better. But who's incentivized to make them all work together better, and allow you to do more things with them together? There's nobody that's really incentivized to do that.”¹⁷⁵

175

Systems Live, Episode 51:
Megalith <http://systemslive.org/>

Timothy Fitz, a systems engineer, further explained:

“Bruce Schneier described this sort of thing as serfdom. So, we're in a world where Google is a city-state, and Apple is a city-state,

and...if I just continue to use Google products, and I stay within their walls, I get this great benefit. [But] living in a mixed world is almost impossible, and very painful, and everything has bugs, and no one of these companies really wants to support you. And so we're in this weird world where, and if you look at a city-state world, one of the big problems was interstate commerce, if you have a tariff because you're trying to export something from Austin and sell it to Dallas, that is not a good economy. You're going to suffer from a lack of innovation and a lack of idea sharing. And that's where we are right now.^{176 177}

176

Systems Live, Episode 51: Megalith <http://systemslive.org/>

177

<http://www.wired.com/2012/11/feudal-security/>

Although the “serfdom” argument mostly refers to a company’s products, such as an iPhone versus an Android, it could also hold true for sponsored open source projects. The first improvements to be prioritized will be those that directly benefit a developer’s employer. This observation is not malicious or conspiratorial, but merely a constraint of being paid by a company to work on a project that is not directly their business.

But nobody can control the origin of a successful open source project, whether Google, the Linux Foundation, or an independent group of developers. New and valuable projects can come from anywhere, and when they provide a valuable service to other developers, they become popular. This is a good thing and fosters innovation.

Dedicated foundation support

Two foundations recently stepped forward to focus more specifically on supporting digital infrastructure: the Linux Foundation and the Mozilla Foundation.

Following Heartbleed, the Linux Foundation announced the formation of the Core Infrastructure Initiative (CII) in order to prevent a similar situation from happening again. Jim Zemlin, the executive director of the Linux Foundation, gathered nearly \$4 million in pledges from thirteen corporate donors, including Amazon Web Services, IBM, and Microsoft, to support security-related infrastructure projects over the next three years.¹⁷⁸ They are also building government support, including support from the White House.¹⁷⁹ The CII is officially a project of the Linux Foundation. Since its formation in April 2014, the CII has sponsored development work on a number of projects, including OpenSSL, NTP, GnuPG (a communication encryption system), and OpenSSH (a set of security-related protocols). The CII primarily focuses on security-related projects as a subset of infrastructure.

178

<http://www.cnet.com/news/tech-titans-join-forces-to-stop-the-next-heartbleed/>

179

<http://www.linuxfoundation.org/news-media/blogs/browse/2016/02/linux-foundation-s-core-infrastructure-initiative-working-white>

In October 2015, Mitchell Baker, chair of the Mozilla Foundation, announced the Mozilla Open Source Support Program (MOSS), pledging \$1M to support free and open source software. According to Baker, the program will consist of a “give back” element for projects that Mozilla relies upon, and a “give forward” element to support free and open source projects at large. However, the first set of grants focused exclusively on the former. Mozilla identified nine projects for its initial set of grants, crowdsourcing suggestions from the community.¹⁸⁰ They have also expressed interest in funding security audits of critical open source projects.¹⁸¹

180

<https://blog.mozilla.org/blog/2015/10/23/mozilla-launches-open-source-support-program/>

181

https://wiki.mozilla.org/MOSS/Secure_Open_Source

Finally, some foundations have made ad hoc contributions to software-related projects. For example, the Python Software Foundation makes small grants to organizations and individuals, mostly related to outreach and education.¹⁸²

182

<https://www.python.org/psf/records/board/resolutions/>

Other institutional actors

There are several remaining actors who lend support to digital infrastructure in various ways: GitHub, venture capital, and academia.

If Facebook is a “social utility”¹⁸³ and Google is a “search utility”—both de facto governing bodies in their respective fields—then GitHub stands a chance to become the “open source utility.” Its business model likely precludes it from ever becoming a similar financial juggernaut (Facebook and Google both benefit greatly from advertising models, whereas GitHub monetizes by hosting code for enterprise clients, as well as charging individuals to host code privately), but GitHub is still the place where today’s open source is made and managed.

183

<http://techcrunch.com/2013/09/18/facebook-doesnt-want-to-be-cool/>

GitHub hinted at greater aspirations when it took \$350M in venture capital, despite already being profitable. If GitHub fully embraces its role as a steward of open source, it could have enormous influence on how those projects get supported. For example, it could create better tools to manage open source projects, advocate for certain types of licensing, or help project maintainers effectively manage their communities.

GitHub has faced mounting pressure from project maintainers on these topics, including a “Dear GitHub” open letter written and signed by maintainers, many from the JavaScript community. The letter explained, *“Many of us are frustrated. Those of us who run some of the most popular projects on GitHub feel completely ignored by you.”* It included a list of requests for product improvements that could help them manage their projects more efficiently.¹⁸⁴

184

<https://github.com/dear-github/dear-github>

GitHub itself is struggling with growing pains that have been well-documented in the media. Early on, the company became famous for its flat hierarchy, with no managers or top-down assignments. GitHub employees were given freedom to work on projects of their own choosing.¹⁸⁵ In recent years, as GitHub has grown to nearly 500 employees, the company has shifted its focus to the enterprise side of its business, hiring sales teams and enterprise executives and adopting a more traditional work hierarchy. The transition from a decentralized to more centralized culture has been difficult for GitHub: at least 10 executives left within a few months spanning the winter of 2015-2016, including the VP engineering, CFO, strategy VP, and human resources VP.¹⁸⁶ Given these internal conflicts, GitHub has yet to publicly signal that it will take an advocacy role and provide leadership around open source's more pressing infrastructure issues, but the potential is there.

185

<http://www.fastcompany.com/3020181/open-company/inside-githubs-super-lean-management-strategy-and-how-it-drives-innovation>

186

<http://www.businessinsider.com/github-the-full-inside-story-2016-2>

Venture capital, as discussed, has a personal stake in the future of digital infrastructure. Because developer tools help technology companies build faster and smarter, the better the tools, the better the startups, and the more venture capital thrives. However, “infrastructure,” from a venture capitalist’s mindset, is not limited to open source but rather focused on platforms that help other people create. Therefore, investments in GitHub or npm, which are platforms that help distribute open source code, make sense, but so do investments like Slack, a workplace collaboration platform which developers can use to create other “command”-driven apps. (To this point, venture capitalists formed a \$80M “Slack fund” to support developer projects that use Slack.¹⁸⁷) Even if venture capitalists appreciate the underlying mechanics of infrastructure, they are limited by their asset class: a VC could not make investments into a project that didn’t have a business model.

187

<http://fortune.com/2015/12/15/slack-app-investment-fund/>

Finally, academic institutions have played a prominent historic role in supporting digital infrastructure, especially the development of new projects. For example, LLVM, a compiler project for C and C++ languages, started as a research project at the University of Illinois at Urbana-Champaign. It is now used in Apple's Mac OS X and iOS development tools, as well as Sony's PS4 development kit.

In another example, R, a popular programming language for statistical computing and data analysis, was initially written by Robert Gentleman and Ross Ihaka at the University of Auckland.¹⁸⁸ R is used not just by software companies like Facebook or Google, but also by Bank of America, the Food and Drug Administration, and the National Weather Service, among others.¹⁸⁹

188

<https://www.r-project.org/contributors.html>

189

<http://www.revolutionanalytics.com/companies-using-r>

Some universities also employ maintainers, who then have the freedom to work on open source projects. For example, Network Time Protocol, used to synchronize time on the Web, was first developed by David Mills, now an emeritus professor at the University of Delaware. (The project continues to be maintained by a group of volunteers, led by Harlan Stenn.) Bash, the aforementioned developer tool, is currently maintained by Chet Ramey, who is employed by the Information Technology Services division of Case Western University.

Academic institutions have the potential to play an important role in supporting newer projects, due to their endowment model and mission alignment, but they can also lack the speed necessary to appeal to modern open source developers. NumFOCUS is an example of a 501(c)(3) foundation that supports open source scientific software through fiscal sponsorship and donations.¹⁹⁰ An external foundation model could help provide the support that

190

<http://www.numfocus.org/open-source-projects.html>

scientific software needs within the context of an academic environment. The Alfred P. Sloan Foundation and the Gordon and Betty Moore Foundation are also experimenting with ways to connect academic institutions with maintainers of data science software, in order to facilitate an open and sustainable ecosystem.¹⁹¹

191

<http://msdse.org/themes/#tools>

Opportunities Ahead

"There is a need for support for the free public infrastructure....People scream if their clocks are off by a second. They say, 'Yes, we need you, but we can't give you any money.'"

- Harlan Stenn, Network Time Protocol

Developing effective support strategies

Although there is growing interest in efforts to support digital infrastructure, current initiatives are still new, ad hoc or provide only partial support (such as fiscal sponsorship).

Developing effective support strategies requires a nuanced understanding of the open source culture that characterizes so much of our digital infrastructure, as well as recognizing that much has changed in the past five years, including the very definition of “open source” itself.

Money alone will not fix a struggling infrastructure project, because open source thrives on human rather than financial resources. There are many ways to grow human resources, such as distributing the workload among more contributors or encouraging companies to make open source part of their employees’ work. An effective support strategy must include multiple ways to generate time and resources besides directly financing development. It must start from the principle that the open source approach is not inherently flawed, but rather under-resourced.

Supporting infrastructure requires embracing the concept of stewardship rather than control. As we’ve seen, digital infrastructure doesn’t look like physical infrastructure. It is distributed across multiple actors and organizations, with projects of many shapes and sizes, and it is hard to predict which projects will be successful or who will contribute to them in the long term. With this in mind, here are some suggested design principles for effective support strategies:

Embrace, rather than fight against, decentralization. Open source is meant to be distributed; that's part of what makes it so impactful. Leverage the community-driven approach as a strength rather than centralizing authority.

Work closely with existing software communities. Software communities are active, tight-knit, and vocal. Treat them as an asset rather than making decisions behind closed doors. Prominent community voices are canaries in the coal mine when something needs attending to.

Consider a holistic approach to project support. Projects need more than just code or money, and sometimes, they need neither. Long-term support is more about creating time than it is about money. Code reviews, technical documentation, code testing, community advocacy, and evangelism are all important resources.

Help project maintainers plan ahead. Current efforts to support digital infrastructure tend to be reactive and ad hoc. In addition to existing projects, there may be new projects that need to be supported and built. For existing projects, maintainers will benefit greatly from being able to plan for the next three to five years, not just six months to a year.

Recognize opportunities, not just risks. Modern open source support is not just about preventing worst-case scenarios (for example, security breaches), but rather empowering more people to build more things. This concept is a hallmark of today's open source culture and also helps build a legacy of support. Consider how you can include more people from different backgrounds, skill sets, and abilities in your strategy, rather than limiting work to benefitting existing participants.

David Heinemeier Hansson, the creator of Ruby on Rails, compared open source to a coral reef:

*It's more sensitive than you think, and it's [hard] to underestimate the beauty that's unwittingly at stake. Please tread with care.*¹⁹²

192

<http://david.heinemeierhansson.com/2013/the-perils-of-mixing-open-source-and-money.html>

Priming the landscape

It is too early to say what long-term institutional support should look like from a programmatic perspective, but there are several critical areas of work that would help us get there.

The following suggestions fall into three areas:

Treating digital infrastructure as a necessary public good and elevating its importance to key stakeholders across sectors

Working with projects to improve standards, security, and workflows

Expanding the pool of contributors so that more people, and more types of people, can build and sustain public software together

Building awareness and educating key stakeholders

As discussed in this report, many key stakeholders—including those from startups, government, and venture capital—mistakenly believe that public software “just works” and does not require additional support. In order to adequately support our digital infrastructure ecosystem, these populations must first be made aware of the problem. Digital infrastructure needs advocates, unhampered by political or commercial constraints, who can understand and communicate the needs of digital infrastructure.

Treating digital infrastructure as a necessary public good could also help direct investment into building better systems from scratch. For example, in the United States, the interstate highway system and the public library system were intentionally designed as public resources. Both had champions (President Dwight Eisenhower and philanthropist Andrew Carnegie, respectively), who built a case for the social and financial benefit that would result from these projects. A national highway system not only connected us as people, making it easier to get from place to place, but brought financial prosperity to all corners of the country, due to commercial use of highways to transport goods. Andrew Carnegie's free public libraries used an "open stack" instead of a "closed stack" system, enabling people to browse and find information themselves, instead of requesting it from a librarian. This practice helped democratize information and empower people to educate themselves.

Better education and awareness could also extend to governments, some of which have made digital infrastructure legally difficult to support, and who may not be as familiar with the cultural norms and history of open source. In the United States, the IRS has narrow definitions of what it considers to be charitable work, and because open source is not well understood, its positive impact on society goes unnoticed. This makes it difficult to institutionalize bigger projects under a foundation or trade association.

Measuring the usage and impact of digital infrastructure

The impact of digital infrastructure is still very difficult to measure. Usage metrics are either highly inaccurate or simply unavailable. This is not an easy problem to solve for. But without data about

which tools are used, and how much we rely upon them, it is hard to paint a clear picture of what is underfunded.

With better metrics, we could describe the economic impact of digital infrastructure, identify critical projects that are lacking support, and understand dependencies between projects and people. Right now, it is impossible to say who is using an open source project unless that person or company discloses their usage. Our information about which projects need better support is mostly anecdotal.

Better metrics could also help us identify “keystone contributors” to open source. In conservation biology, a “keystone species” is a species of animal with a disproportionately large effect on its environment relative to its abundance.¹⁹³ Similarly, a “keystone contributor” might be a developer who contributes to multiple critical projects, is singlehandedly responsible for a critical project, or is generally perceived to be influential and trustworthy. Keystone contributors are critical advocates; empowering them with the resources they need could help improve the system as a whole. Understanding the relationship between open source communities and keystone contributors could help quickly identify areas that require further support.

There is also little data about the contributors themselves: who contributes to open source, what conditions allow them to do so, and what types of contributions are made. Women, non-English speakers, and new contributors to open source are examples of demographics that should be tracked over time, especially to measure the impact of support programs.

193

https://en.wikipedia.org/wiki/Keystone_species

The only statistics available about GitHub repositories are the number of people who have starred (similar to a “like” or “favorite”), watched (meaning they receive updates about the project) or forked a project. These numbers help provide some metrics for relative popularity, but can also be misleading. Plenty of people could star a project, for example, because it is conceptually interesting, but not actually use it in their code.

Some package managers like npm (which is used for Node.js) track downloads. Debian Popularity Contest tracks downloads of packages related to the free operating system Debian. Each package manager is limited to a particular ecosystem, however, and no one package manager can paint a picture of the system at large. Many projects are not part of a package manager and go untracked. Libraries.io, a website created by Andrew Nesbitt, is one effort to aggregate data open source projects and provide more data around their usage; it tracks over 1.3M open source libraries across 32 package managers.¹⁹⁴

194

<https://libraries.io/>

Working with projects to modernize workflows

Many projects struggle not just due to lack of funding, but because the projects are difficult to contribute to, or suffer a bottleneck from maintainers, who meticulously review and accept pull requests from the community. This is particularly true for older projects which may have been built using developer tools, languages, or workflows that are no longer popular (for example, using an older version control system instead of Git, whose popularity is growing among developers).

There is plenty of work that can be done to make projects easier to contribute to, including migrating them to newer workflows, cleaning up code, closing unattended pull requests, and setting clear policies for contribution.

Some projects have experimented with making it easier to contribute. For example, developer Felix Geisendörfer has suggested that everybody who submits a change to code should be given commit access, in order to reduce the bottleneck of a single maintainer reviewing and approving those changes. Felix found that *“this approach is a fantastic way to keep projects from going stale as well as turning one-man projects into small communities.”*¹⁹⁵

195

<http://felixge.de/2013/03/11/the-pull-request-hack.html>

The Node.js contribution policy, which is made available for other Node projects to adopt, emphasizes growing the number of contributors and empowering them to make their own decisions, instead of treating maintainers as the final approving authority. Their contribution policy details how to submit and accept pull requests and how to log bugs and other issues. The Node.js maintainers found that adopting better policies helped them manage their workload and grow their community into a healthier, active project.¹⁹⁶

196

<https://medium.com/the-javascript-collection/healthy-open-source-967fa8be7951#4x37jao9w>

There is research to be done that addresses what projects should strive towards in the first place. That is, what does a “successful” project look like, in terms of financial support and governance models, as well as balancing the right mix of maintainers, contributors, and users? The answer may vary for different types or sizes of projects.

Encouraging standards across open source projects

Although GitHub is becoming a standard platform for code collaboration, many aspects of an open source project are still not standardized, including the breadth and depth of documentation, licenses, and contributing guides, as well as code style and formatting. Encouraging the adoption of project standards can make it easier for maintainers to manage contributions, as well as lowering a contributor's barrier to participation.

One example of a growing standard is a code of conduct, which is a policy detailing expectations for behavior and communication. Codes of conduct are being adopted among many project communities in recent years, including Node.js, Django, and Ruby. Although the process of adoption has been hotly debated among some communities, their proliferation suggests a rising interest in holding communities accountable for their behavior.

Expanding the pool of open source contributors

As discussed earlier in this report, software is a booming industry, with growing numbers of just new developers but other skilled talent, and there is work to be done to encourage newcomers to contribute to open source. Expanding the pool of contributors helps open source projects become more resilient, because more people are participating in their development. Helping more people contribute to open source also increases empathy and communication between open source “users” and the projects they depend on.

Your First PR is an example of an initiative, developed by programmer Charlotte Spencer, that helps newcomers make their first contribution to open source.¹⁹⁷ First Timers Only¹⁹⁸ and Make a Pull Request¹⁹⁹ are two other popular examples of resources that introduce newcomers to open source. Some open source projects also use tags such as “first bug” or “contributor friendly” to flag issues that are suitable for less experienced contributors to tackle. It would also be valuable to encourage contributions to open source beyond code, such as writing technical documentation, managing tasks and workflows, or creating a website for a project.

197

<https://twitter.com/yourfirstpr>

198

<http://www.firsttimersonly.com/>

199

<http://makeapullrequest.com/>

In addition to increasing the percentage of technical talent that contributes to open source, there is an opportunity to draw from a wider pool of contributors. Making non-English speakers feel welcomed in open source communities, for example, can help make technology more accessible around the world. And because many recruiters use open source work as a portfolio when hiring developers, a more diverse open source community can help build a more inclusive tech talent field overall.

Improving relationships between projects and external stakeholders

Companies are an inevitable part of the open source ecosystem, and their role is only increasing in importance as more companies embrace open source software. Making it easier for companies and projects to work with one another, as well as helping companies understand the needs of project communities, can unlock companies as patrons of, and advocates for, open source.

According to the annual Black Duck open source company survey,

only 27% of companies have a formal policy for employee contributions to open source.²⁰⁰ Clarifying whether and how employees may contribute back to open source on company time, and encouraging them to do so, could go a long way in improving corporate support for open source projects.

200

<https://www.blackducksoftware.com/news/releases/seventy-eight-percent-companies-run-open-source-yet-many-lack-formal-policies-manage>

In 2014, a group of companies formed the TODO Group, to share best practices around corporate participation in open source. Members include Box, Facebook, Dropbox, Twitter, and Stripe.²⁰¹ In March 2016, the TODO Group announced it would be housed by the Linux Foundation as a collaborative project.²⁰²

201

<http://todogroup.org/>

202

<http://todogroup.org/blog/todo-becomes-lf-collaborative-project/>

Companies can also provide financial support for projects, but sometimes find it difficult to figure out how to structure their sponsorship. Creating sponsorship budgets for engineering departments or employees, or creating documents to make it easy for projects to “invoice” companies, could increase financial contributions to open source.

Poul-Henning Kamp, for example, works on an open source project called Varnish, used by one-tenth of the top websites on the Internet, including Facebook, Twitter, Tumblr, The New York Times, and The Guardian.²⁰³ To fund his work, he created the Varnish Moral License to make it easy for companies to sponsor the project. Although in practice the relationship is a sponsorship, Poul-Henning uses terminology that companies are familiar with, such as “invoices” and “licenses,” to reduce barriers to participation.²⁰⁴

203

[https://en.wikipedia.org/wiki/Varnish_\(software\)](https://en.wikipedia.org/wiki/Varnish_(software))

204

<http://phk.freebsd.dk/VML/>

Increasing support of diverse skill sets and non-coding functions

In the not-too-distant past, software startups were once heavily weighted towards engineering talent. Other functions, like marketing or design, were considered secondary to code.

With the rapid creation and consumerization of software today, that view no longer holds. Startups need to compete for their customers' attention. Brand has become one of the most important differentiators.

The last five years have seen the rise of the “full stack engineer”: developers who are more generalists than specialists, able to work on different layers of software complexity, and who might even have some proficiency with design or product. Software teams collaborate more closely, using agile software development approaches (where the product is built through frequent iteration between engineering, design, product, and marketing teams) rather than waterfall approaches (where each team completes their piece of the product before handing it to the next team).

Open source software has seen very few of these changes, despite our increasing reliance on these projects. Understandably, code is central to an open source project, since in some ways it is the “product” or output. Less valued are functions such as community, documentation, or evangelism, that are the mark of any healthy, sustainable organization. As a result, projects become imbalanced.

There is plenty of work that could be done to fund and support non-code contributions, in-kind donations (such as paying for servers), and benefits (such as health insurance). Having this type of support could go a long way in easing the burden on maintainers.

The crossroads we face

The current state of our digital infrastructure is one of the most poorly understood issues of our time. It is critical that we understand it.

By making a voluntary investment in our underlying infrastructure, developers made it easier for others to build software. By giving it away for free instead of charging for it, they fueled an information revolution.

Developers did not do this for altruistic reasons. They did it because it was the best way to solve their own problems. The story of open source software is one of the great modern day triumphs of the public good.

We are lucky that developers have borne the hidden cost of these investments. But their initial investments only get us so far.

We are merely at the beginning of the story of how software transformed humanity. Marc Andreessen, the co-founder of Netscape and well-known venture capitalist behind the firm Andreessen Horowitz, observed in 2011 that “*software is eating the world.*”²⁰⁵ Since then, that statement that has become canon for the modern age.

Software affects everything we do: not just the frivolous and entertaining, but the mandatory and critical. OpenSSL, the project described at the beginning of this paper, demonstrates this well. In a phone interview, Steve Marquess explained that OpenSSL was used not just by consumer websites, but by the government, drones, satellites, “*any gadget you hear in the hospital beeping.*”²⁰⁶

205

<http://www.wsj.com/articles/SB10001424053111903480904576512250915629460>

206

Phone interview with Steve Marquess

The Network Time Protocol, maintained by Harlan Stenn, synchronizes the clocks used by billions of networked devices and affects everything with a timestamp: not just messaging apps or email, but financial markets, medical records, and chemical processing.

And yet, Harlan observes:

*“There is a need for support for the free public infrastructure. But there's just no revenue stream around time right now. People scream if their clocks are off by a second. They say, ‘Yes, we need you, but we can't give you any money.’”*²⁰⁷

207

<http://www.informationweek.com/it-life/ntp-fate-hinges-on-father-time/d/d-id/1319432>

In the last five years, open source infrastructure has become an essential layer of our social fabric. But much like startups or technology itself, what worked for the first 30 years of open source's history won't work moving forward. In order to maintain our pace of progress, we need to invest back into the tools that help us build bigger and better things.

Figuring out how to support digital infrastructure may seem daunting, but there are plenty of reasons to see the road ahead as an opportunity.

Firstly, the infrastructure is already there, with clearly demonstrated present value. This report does not propose to invest in an idea with unknown future value. The enormous social contributions of today's digital infrastructure cannot be ignored or argued away, as has happened with other, equally important debates about data and privacy, net neutrality, or private versus public interests. This makes it easier to shift the conversation to solutions.

Secondly, there are already engaged, thriving open source communities to work with. Many developers identify with the programming language they use (such as Python or JavaScript), the function they provide (such as data science or devops), or a prominent project (such as Node.js or Rails). These are strong, vocal, and enthusiastic communities.

The builders of our digital infrastructure are connected to each other, aware of their needs, and technically talented. They already built our city; we just need to help keep the lights on so they can continue doing what they do best.

Infrastructure, whether physical or digital, is not easy to understand, and its effects are not always visible, but this should compel us to look more, not less, closely. When a community has spoken so vocally and so often about its needs, all we need to do is listen.



Appendix



Glossary

Code repository: The location of source code needed to use a software project. For example, GitHub offers a place to host one's repository so that other people can find and use it. Also colloquially called a "repo." The collection of source code files themselves are called the "codebase."

Digital infrastructure: For the purposes of this report, digital infrastructure refers to public software components that are used to build software for personal or commercial use. Examples include programming languages and databases. This definition does not include physical infrastructure needed to build software (e.g., physical servers or cables).

Fork: There are two types of forks. A **project fork** is the historical definition of a fork, in which someone makes a copy of an open source project and continues to develop it separately. It is used politically; for example, when there is internal disagreement about the project's direction, or to force substantial changes to the original project (ideally, merging the fork back into the main project). A **GitHub fork** refers to temporarily copying a project to make changes, usually with the intent of merging those changes back into the main project. It does not carry the political and social weight of a project fork. GitHub repurposed this term to encourage a culture of lightweight "tinkering" that is now prevalent among modern open source contributors.

Contributor (open source): Someone who has made a contribution to a live open source project. Examples of contributions include writing code, documentation, or managing support issues. A contributor may not have commit access to a project (i.e. someone else must approve their contributions before they are live).

Documentation (software): Written information that explains how people can use or contribute back to a software project. Documentation is like an instruction manual for software. Without it, a developer wouldn't know how to use the project.

FOSS, FLOSS, OSS: FOSS is an acronym that means Free and Open Source Software. It is meant to be inclusive of both terms that refer to public software. FLOSS refers to Free, Libre, and Open Source Software and is arguably the most inclusive definition. (The Spanish word libre—"free" as in freedom—is used to distinguish from gratis - "free" as in cost—in order to highlight that "free software" refers to the former.) OSS refers to Open Source Software only.

Free software: Software that is free to run for any purpose (commercial or non-commercial) as well as be studied, changed, and distributed. The term originated around 1983 from the work of computer scientist Richard Stallman, the GNU Project, and Free Software Foundation (founded 1985).

GitHub: A commercial platform for hosting code. GitHub launched in 2008 and is currently the most popular platform for people to host and collaborate on open source projects. (One can also host private code on GitHub.) GitHub helped standardize open source development practices and bring open source to a wider audience. Projects on GitHub use the version control system Git.

Open source software: Open source software has the same technical definition as free software (see above). However, culturally, open source tends to highlight the pragmatic benefits of public software, whereas free software is a social movement. The term “open source” originated from a 1998 meeting of parties interested in exploring a corporate friendly alternative to the term “free software.”

Software framework: Software frameworks provide basic scaffolding for an application. Think of it as a blueprint. Like a blueprint, a framework lays out how the application might look on mobile, or how information gets saved into the database. Examples include Rails and Django.

Source code: For the purposes of this report, source code is the actual code associated with an open source project.

Maintainer (open source): Someone who assumes the responsibility of an open source project. The definition varies from project to project. Sometimes maintainers are formally named in a project, and sometimes they emerge de facto based on who is doing the bulk of the work. A maintainer likely carries the burden of holistic project management more than any individual contributor. They may or may not have authored the original version of the project. They will likely have commit access to a project (i.e. can make changes directly to the project).

Programming language: Programming languages are the communication backbone of software. They help different software components perform actions and talk to one another. Popular examples of languages include JavaScript, Python and C.

Software library: Software libraries are “prefabricated” pieces of code that make it faster to write software, just as a construction company might buy prefabricated windows instead of building them from scratch. For example, instead of a developer writing their own user login system for an application, they can use a library called OAuth.

Venture capital: A type of private equity that provides money to early-stage, high-growth companies in exchange for equity. Venture capital helped grow many commercial aspects of the Internet and gave rise to Silicon Valley.

Acknowledgements

Thanks to everybody who bravely agreed to be cited in this report, as well as to those whose frank and thoughtful perspectives helped round out my thoughts during the research process: André Arko, Brian Behlendorf, Adam Benayoun, Juan Benet, Cory Benfield, Kris Borchers, John Edgar, Maciej Fijalkowski, Karl Fogel, Brian Ford, Sue Graves, Eric Holscher, Brandon Keepers, Russell Keith-Magee, Kyle Kemp, Jan Lehnardt, Jessica Lord, Steve Marquess, Karissa McKelvey, Heather Meeker, Philip Neustrom, Max Ogden, Arash Payan, Stormy Peters, Andrey Petrov, Peter Rabbitson, Mikeal Rogers, Hynek Schlawack, Boaz Sender, Quinn Slack, Chris Soghoian, Charlotte Spencer, Harlan Stenn, Diane Tate, Max Veytsman, Christopher Allan Webber, Chad Whitacre, Meredith Whittaker, Doug Wilson.

Thanks to everybody who wrote something public that was referenced in this paper. This was a critical part of the research, and I am thankful to those who make their ideas public so others can learn from them.

Thanks to Franz Nicolay for copy editing and Brave UX for the design of this report.

Finally, a very special thanks to Jenny Toomey and Michael Brennan for driving this project with patience and enthusiasm, to Lori McGlinchey and Freedman Consulting for their early feedback, and to Ethan Zuckerman for making the magic happen.

SPONSORED BY

