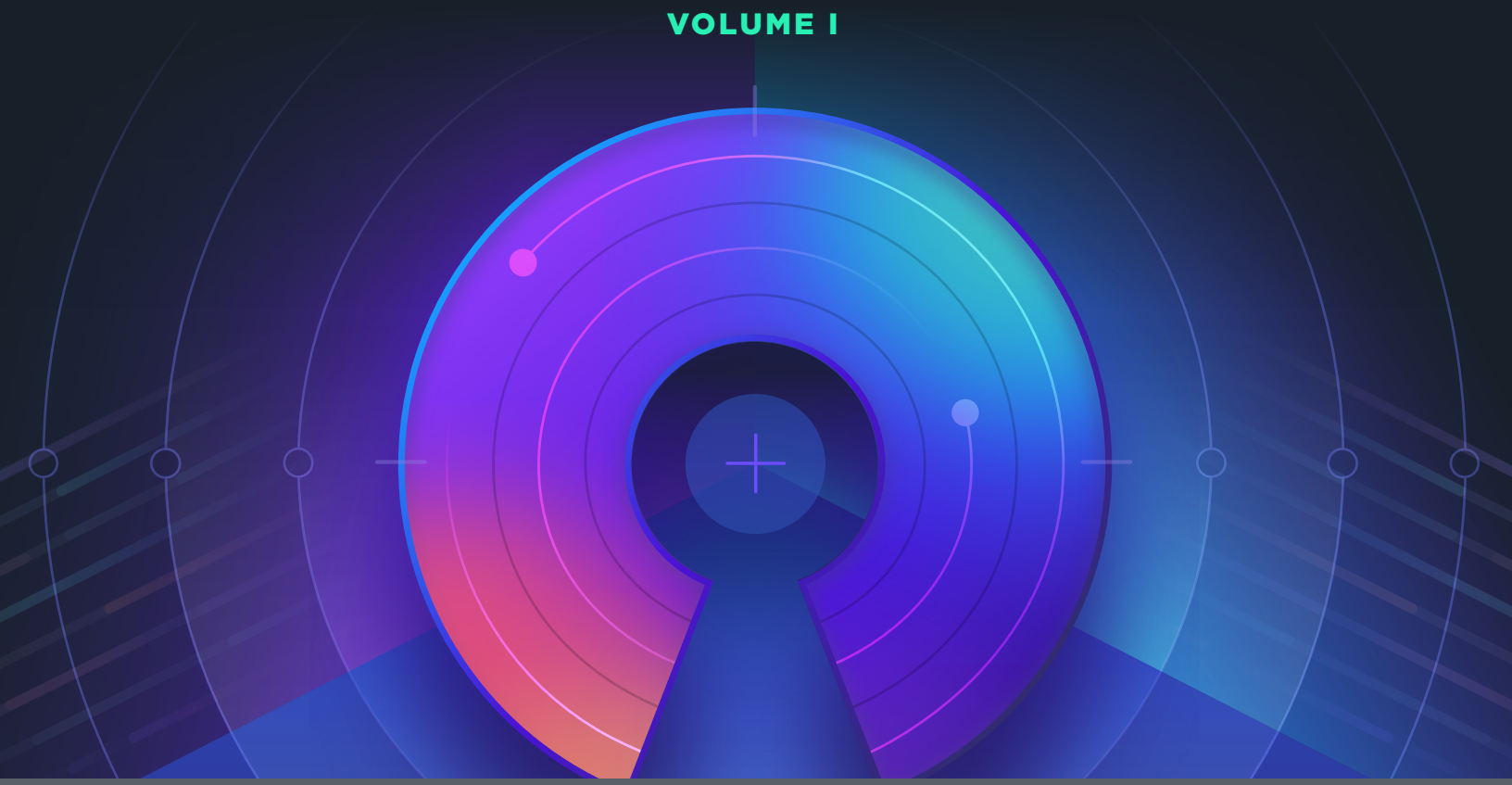
DZone

THE 2018 DZONE GUIDE TO

Open Source

DEMOCRATIZING DEVELOPMENT

VOLUME I



BROUGHT TO YOU IN PARTNERSHIP WITH

Flexera

Dear Reader,

Twenty-two years ago, Eric S. Raymond took over a project called "pop-client" that originated from the work of Carl Harris. The program was to be a software utility for POSIX-compliant operating systems which would be utilized to retrieve e-mail from a remote POP3/IMAP/ETRN/ODMR mail server and store the data onto the user's local system.

Raymond had a vision to finish the development in a manner by which all the source code was available to anyone who wanted to review or contribute to the project. Soon after, the project was given a new name of "fetchmail" and went on to become one of the first open-source and software projects on record.

The larger benefit to this project is that Eric S. Raymond used the fetch-mail endeavor as a springboard to express his theories on open-source software development. His thoughts were proclaimed in his essay "The Cathedral and the Bazaar" — a publication which was ultimately released under the Open Publication License.

Fast-forward to today and the impact open-source software has had on development teams is nothing less than spectacular. This does not only cover traditional web-based applications, but mobile applications being deployed daily as well. Outside the development circle, open-source solutions are a driving force behind DevOps implementations as well, with other shared services started to utilize open-source solutions.

Needless to say, I am excited to write the welcome letter for DZone's Open-Source Guide.

In this guide, we will dive a little deeper into the early history of open-source software development, which should further frame the concepts and philosophies that have become commonplace for most open source initiatives.

Feel like you want to contribute, but you don't know where to start? We've included a primer to help get you started with the GitHub service. Gaining an understanding of the innerworkings of the GitHub remote repository for your first project, can help ease the learning curve when you are ready to contribute to your first open-source collaboration.

Of course, without standards and practices in place, open-source development can quickly start to resemble the "wild west" of software design. That's why we have decided to dedicate a section of this guide to the necessity of community guidelines.

We are also planning to include a case study, focusing on how an open-source software approach helped advance big data technology. I truly believe that the benefits obtained from an open-source approach can be employed across multiple segments of a deep and wide Information Technology landscape.

Enjoy our guide, leverage open-source software where it makes sense, and contribute where your skills and abilities can make a difference to others.

By John Vester

SR. ARCHITECT, CLEANSLATE, AND ZONE LEADER, DZONE

Table of Contents

DZONE IS...

BUSINESS AND PRODUCT
MATT TORMOLLEN
CEO

MATT SCHMIDT
PRESIDENT

JESSE DAVIS
EVP, TECHNOLOGY

KELLET ATKINSON
MEDIA PRODUCT MANAGER

PRODUCTION
CHRIS SMITH
DIR. OF PRODUCTION

ANDRE POWELL
SR. PRODUCTION COORD.

G. RYAN SPAIN
PRODUCTION COORD.

ASHLEY SLATE
DESIGN DIR.

BILLY DAVIS
PRODUCTION ASSISSTANT

JASON BUDDAY
ACCOUNT MGR.

MICHAELA LICARI
ACCOUNT MGR.

EDITORIAL
CAITLIN CANDELMO
DIR. OF CONTENT & COMMUNITY

MATT WERNER
PUBLICATIONS COORD.

SARAH DAVIS
PUBLICATIONS ASSOCIATE

MICHAEL THARRINGTON
CONTENT & COMMUNITY MGR. II

KARA PHELPS
CONTENT & COMMUNITY MGR.

TOM SMITH
RESEARCH ANALYST

MIKE GATES
SR. CONTENT COORD.

JORDAN BAKER
CONTENT COORD.

ANNE MARIE GLEN
CONTENT COORD.

ANDRE LEE-MOYE
CONTENT COORD.

LAUREN FERRELL
CONTENT COORD.

SALES
CHRIS BRUMFIELD
SALES MANAGER

FERAS ABDEL
SALES MANAGER

ALEX CRAFTS
DIR. OF MAJOR ACCOUNTS

JIM HOWARD
SR. ACCOUNT EXECUTIVE

JIM DYER
SR. ACCOUNT EXECUTIVE

ANDREW BARKER
SR. ACCOUNT EXECUTIVE

BRIAN ANDERSON
ACCOUNT EXECUTIVE

SARA CORBIN
ACCOUNT EXECUTIVE

BRETT SAYRE
ACCOUNT EXECUTIVE

MARKETING
LAUREN CURATOLA
MARKETING SPECIALIST

KRISTEN PAGÀN
MARKETING SPECIALIST

JULIAN MORRIS
MARKETING ASSOCIATE

Executive Summary

BY MATT WERNER
PUBLICATIONS COORDINATOR, DZONE

From Linux and GPL to Ethereum and Hadoop, open-source software has been at the forefront of computing and software development for over thirty years. Developers have been freely using, improving, and forking projects to build better software and, in some cases, alter or create entire industries and development processes. For our first-ever Guide to Open Source, we asked 608 DZone members about what projects they used, what drew them to open-source software, and how they're contributing back.

COMMUNITY RULES

Data: The biggest reasons for choosing a specific open-source tool are helpful communities (74%), popularity with developers in general (68%), and the maturity of the solution (62%). Welcoming communities are the third biggest reason why developers like open source (54%), after reducing development costs (80%) and no vendor lock-in (59%).

Implications: Welcoming, helpful communities are one of the best things an open-source project can have. The more potential users feel they can start using and contributing, the more adoption that project will have.

Recommendations: Take the time to answer questions in a respectful, helpful manner, and ask for help in fixing bugs or getting user input into new features. Create a code of conduct that clearly states what kind of behavior is and isn't acceptable, using terms that encourage the kind of behavior you want to see rather than simply prohibiting what you don't. Lisa Smith's article on page 18 goes into more detail about the importance of community guidelines.

HOW DZONE CONTRIBUTES

Data: 84% of DZone members do not contribute to open-source projects. Those that do contribute by fixing bugs (68%), answering questions for new users (59%), and adding new features (53%). 44% of survey respondents are "somewhat likely" to contribute in the future and only 6% were "not at all likely."

Implications: There will always be more adopters of open-source projects than there are contributors, but there could still be more contributors. Those who do contribute are more likely to improve currently existing features than to try and add new ones.

Recommendations: Think of what bugs you encounter using one of your open-source tools and try to fix the problem yourself. You can do this quite easily though GitHub and can help you establish yourself within the community as someone who can and is willing to help out.

THE DARK SIDE OF THE OSS

Data: 62% of respondents said their company has no formal policy for selecting or integrating open-source software. In addition, the biggest challenges of adopting open source were unclear documentation (67%), lack of documentation (64%), and the lack of external resources (such as tutorials or presentations on third-party sites) (45%).

Implications: According to the survey results, several businesses are amassing severe amounts of technical debt. When these businesses do need to refactor or update their software, it will actually add money to development costs since there's little to no visibility into what tools are being used in development. Additionally, this further highlights the importance of community-provided documentation efforts.

Recommendations: If you use open-source projects and find their documentation obtuse or sparse, or can't find the documentation at all, try your hand at writing your own, and ask for community feedback on whether it's better or worse than the current available resources. You can also spread the word about your favorite project by writing tutorials or use cases on a blog or by sharing them on sites like DZone. IT leaders in businesses need to adopt a formal policy for adopting open source as soon as possible before costs rise even more.

Key Research Findings

BY JORDAN BAKER
CONTENT COORDINATOR, DZONE

DEMOGRAPHICS

629 software professionals completed DZone's 2018 Open Source Survey. The following is a snapshot of the respondents' demographics:

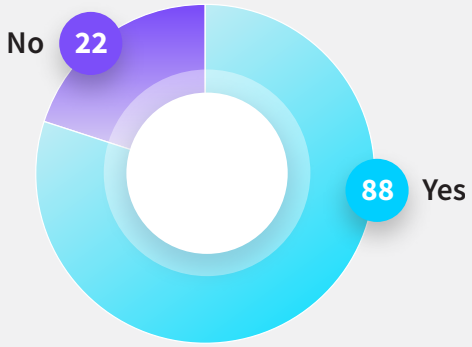
- 34% identify as developers/engineers, 19% as developer team leads, and 14% as architects.
- As their primary language at work, 60% use Java, 8% use JavaScript/Node.js, and 8% use Python.
- 82% work at companies that use the Java ecosystem; 68% work for companies that use a JavaScript ecosystem on the client-side; 40% use the Python ecosystem at work; 36% employ Node.js as their server-side ecosystem at work.
- 6% work for companies headquartered in Europe, 25% in the USA, and 18% in South and Eastern Asia.
- 23% work for companies with 100-999 employees, 19% for companies with 1,000-9,999 employees, and 21% for companies with 10,000+ employees.
- 20% work for a software vendor, 17% in the finance/banking industry, 11% in consulting, and 10% for e-commerce/internet-based organizations.

OPEN SOURCE VS. PROPRIETARY SOFTWARE

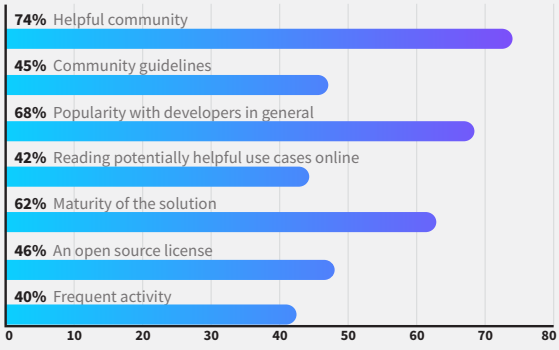
Before diving into how and why the DZone community uses open-source software, let's take a quick look at how open-source and proprietary software stack up in terms of popularity among the developer community. In the research survey for this Guide, we asked respondents four basic questions regarding the open source vs. proprietary debate that dealt with the security, stability, and ease of use of either solution, as well as whether they prefer to look for OSS or proprietary when planning a new project.

The greatest level of parity when comparing proprietary and open source came when we asked developers to tell us which they found more secure. 47% of the 629 respondents claimed they felt that, when it comes to security concerns, open-source and proprietary solutions are "about the same." Interestingly, only 18% of

GRAPH 01. Do you use open source projects in your job?



GRAPH 02. Which factors influence your decision to use a particular open-source project?



respondents said that they felt proprietary software had definitively better built-in security protocols than open-source software.

When asked to compare the stability of OSS and proprietary solutions, our community again showed a large sense of parity between the two categories. 42% of the 629 respondents told us that they see an equal level of stability between open-source and propriety software. The remaining 58% split more evenly than in the previous question about security. 32% claimed they felt open source is definitively more stable, while 26% expressed the same sentiment for proprietary software.

That's where any sense of equality between open-source and propriety solutions ends for our community, however. When asked which type of software is easier to use, 50% of respondents chose open source, while only 20% said they prefer proprietary solutions' ease of use. Given the large, and, often times, helpful communities that exist around the major open-source initiatives, it makes sense that so many developers prefer open source; though, the disparity between the two is a bit surprising.

Finally, when asked whether they prefer proprietary or open source when selecting software for projects, an overwhelming majority (76%) told us that they prefer open source. Taking the above three questions into consideration, the overall popularity of open source should not be a surprise (though the shear inequality of the responses here, with proprietary only garnering 5% of the vote, is shocking). Here, we see that fact that most respondents told us that, in terms of security and stability, the two solutions were equal, and open source is easier to use, culminating in OSS's extreme popularity among our community.

So, now that we've established the popularity of open source as compared to proprietary software, let's examine in more depth why the DZone community uses OSS in their projects, both at home and at work.

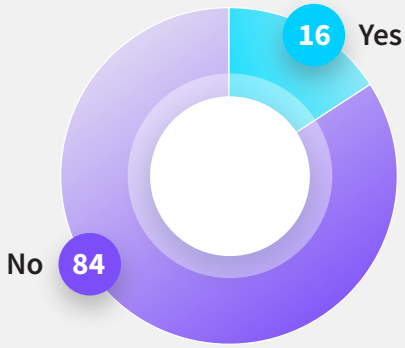
OSS IN PERSONAL AND PROFESSIONAL PROJECTS

Among our respondents, 69% use open-source software in their personal projects and 88% use it at work. That's a considerable number. To get a more detailed picture of why and how developers use open-source software, let's explore how our community, both at home and the office, interacts with OSS solutions and the communities around them.

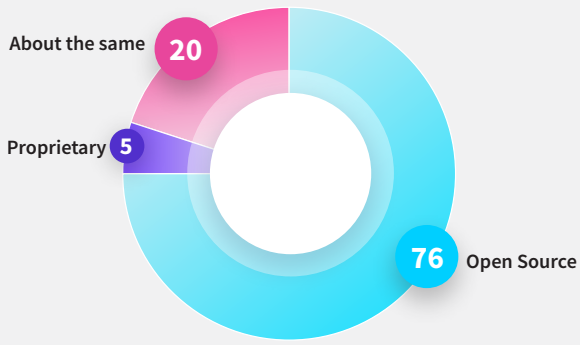
When development organizations are looking to choose an open-source solution, the most important factors all dealt with organizational and development efficiency. The top concern respondents' companies take into account are cost savings (69%), development time (64%), time to integrate into existing software (64%), and security (50%). Interestingly, however, when asked on an individual level what factors influence their choice of OSS for a work project, developers told us: reduction of development costs (20%), no vendor lock-in (15%), and a welcoming community (13%). It thus appears that community interactions are far more important for individual developers than the organizations for which they work. And, when asked what makes OSS an attractive choice for their personal projects, respondents' answers were identical to their professional reasons for choosing OSS: reduction in development costs (19%); no vendor lock-in (14%); welcoming communities (14%).

So, developers use open-source solutions for much the same reasons at work as they do in their spare time. But do the challenges they face in their use of OSS on the job compare to the challenges they encounter while hacking away at home?

GRAPH 03. Do you contribute to open source projects?



GRAPH 04. Which do you prefer to look for when selecting software for your projects?



Those respondents who told us that they use open source at work reported three main challenges that face they consistently: unclear documentation (27%); lack of documentation (25%); lack of outside resources (17%). Thus, the main professional concerns of individual, OSS-based developers are all community-centric. When we asked developers working on personal projects this same question, the results were almost a mirror image. Those respondents who use open source in personal projects reported four main challenges: unclear documentation (26%); lack of documentation (25%); lack of outside resources (17%); hostile community (12%).

While it may seem a matter of common sense that developers use OSS in personal and professional projects for the same reasons, it is nonetheless interesting to note how important a role the open-source community plays in both types of projects. Given that, in the following section, we'll examine how developers interact with open source communities.

COMMUNITY

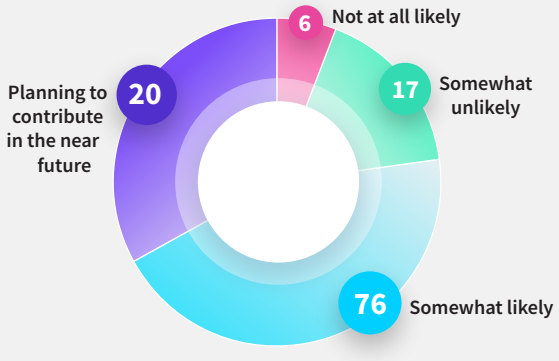
When it comes to the adoption of an open-source solution, the community surrounding it plays just as large a role as the software itself. Indeed, the two biggest reasons respondents expressed for choosing open-source technologies are community-centric in nature. The most popular reason for adopting a particular OSS solution is a helpful community, with 74% of respondents claiming this plays a role in their adoption of the software. Popularity of the solution among developers was the second most popular reason, with 68% of telling us this factor influences their decision to use a particular open-source project. Some other important factors included: maturity of the solution (62%); community guidelines (45%); reading potentially helpful use cases online (42%); frequent activity (40%). Here again, the community-centric nature of a developer's choice of open source projects is apparent.

Though devs are occasionally faced with hostile community interactions, these situations seem to be few and far between. We asked our readers how often they experience negative interactions when dealing with fellow community members, and 65% told us they only have such experiences a few times a year; the second most popular answer was a few times per month, with only 12% of respondents.

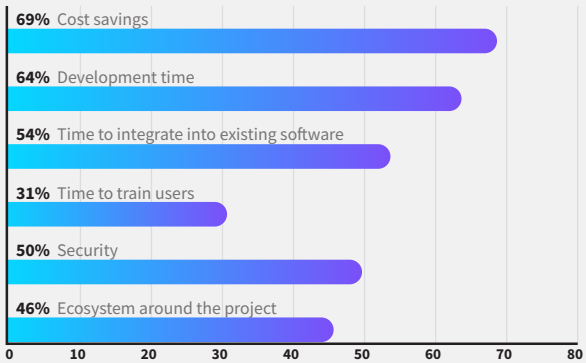
While the community around a particular open-source project will define its adoption rate, most developers do not contribute to open-source projects. Of the 469 respondents who told us that they use open-source in personal projects, 44% said they contribute to open source. And of the 594 developers who said they use open source in their professional projects, only 27% stated that they contribute to open source. Indeed, whereas 88% of total survey respondents use open-source software, only 16% actively contribute to projects. Of those who contribute, the most popular capacity in which they contribute is to help with bug fixes (68%), followed closely by answering questions from new users (59%) and adding new features (53%).

Despite this low contribution rate, only 6% of respondents said that they are not likely to contribute to projects in the future. Of those who use open source in their personal projects, 27% said they are “somewhat likely” to contribute to OSS projects in the future, and 25% said they are “planning to contribute in the near future.” Of those who use open source for projects at work, 38% said they are somewhat likely to contribute to open source at some point down the line, and 30% said they are planning on contributing soon. Thus, as open source continues to gain in popularity, it seems as though the contributor population will continue to increase, as well.

GRAPH 05. How likely are you to contribute to open source in the future?



GRAPH 06. What factors help your company decide whether or not to use open source software?



A Brief History of Open-source

BY STEFAN THORPE
CTO, CAYLENT

QUICK VIEW

01. This history covers the rise of Linux, Apache, and the wealth of open-source projects available today.

02. We look at the impact the open-source philosophy has had on software development.

03. We navigate through Git and what code repositories help teams achieve.

04. And finally, we examine the revolutionary business models that open-sourcing has generated, as well as their drive for a different ethos in programming.

Let's begin with the obvious: open source has revolutionized computing. There are no two ways about it. Seasoned tech heads and rookies alike can now develop and play with code in a way that wasn't possible back in the day. With the help of this innovative approach, computers and software can now be easily adapted to suit individuals, encouraging a collaborative community of creative people helping and developing one another's work.

Do you know where the whole movement started? Well, buckle up as I take you on a brief, yet informative journey through the history of open source. We'll start in 1969, with the rise of Linux and its creation. From Linux, we branch into Apache, the wealth of open-source projects that are available, and the impact both of these elements have had on software development. We'll touch on the growing array of open-source tools now available, including Git, and what code repositories help IT teams achieve. And we'll finish off with a look at how the open-source business model has been a game changer.

In the beginning, there was UNIX — developed at AT&T Bell Labs in the 70s by researchers who had been working on the operating system Multics alongside the Massachusetts Institute of Technology and General Electric. However, frustration around the Multics project grew, and the AT&T Bell Labs group departed the joint project to pursue their own design. UNIX, the blueprint of modern-day open source, was born. The UNIX project bore its own problems, though; after the source code was published for free, many people began to work on their own versions. But, importantly, the seed of collaborative software had been planted.

At this point, it's important to note that you can't write a history of open source without a nod to Richard Stallman. Back in the '80s, Stallman had the idea to create free software that users could adapt when needed. This was reported to be in response to access denial for a laser printer code he requested, which he wanted to modify to make life easier for himself and co-workers. This was Stallman's catalyst for developing a new concept for coding, which he established as the Free Software Foundation. This milestone also marked the beginning of the GNU project. Stallman quit his job and told the world that software should be free to all. Stallman's vision was, and still is, intimately linked with the political and social values of the open-source philosophy.

However, even with all the work Stallman put into rewriting UNIX, GNU was missing something vital. Thankfully, deep in the heart of Finland, a student was working on the missing link.

Let's skip to the 1990s. While everyone was walking around with a Sony Discman and playing Tamagotchi, Linus Torvalds was writing an online post about an ongoing hobby he had. Torvalds was responsible for the "Linux" kernel, the much-needed missing piece for Stallman's work, as it effectively allowed hardware to speak to software. When combined, the GNU/Linux distribution permitted others to begin creating their own operating systems. And now the real fun could begin.

Linux grew stronger and stronger and nowadays, it's hard to find technology that doesn't incorporate it. But open source doesn't stop there.

Enter Apache, a small-time web server created by developers in the early '90s. Continuing the same accessible ethos of Linux, despite Apache's simple start, it is now a significant part of the web. A play on words, "A Patchy Server," it's one of the front-running web servers available, used by 46% of websites. Currently, there are over 350 open-source initiatives on the Apache Software Foundation alone. The ASF also has numerous tools available for content, databases, libraries, and network servers.

Add high security, continuous editing, and innovative upgrades to the collaborative nature of open source, and you have a recipe for success. While some people may argue that the nature of its openness means it is impossible to keep secure, many consider the opposite to be true. Bugs are identified and dealt with quickly and efficiently by an army of global programmers.

One of the issues that arose with editing other people's code happened when changes caused problems to the original software, negatively impacting its function. It became evident that there needed to be a system in place whereby users could take code, alter and test it out offline to ensure it worked correctly, and then merge it once all the bugs had been fixed.

Once again, Torvalds stepped up to deliver a solution. The very nature of open source necessitates a place for easy code access and storage. There were a couple of tools attempting something similar, such as BitKeeper. However, after copyright holder Larry McVoy withdrew free usage rights, many users stopped using BitKeeper, including Torvalds himself. Seeing a gap in the market, Torvalds created Git. Widely considered to be one of the most popular, if not *the* most popular, version control system available today, Git is used across the world. The repository framework enables users to access, edit, and distribute code safely. Git provides somewhere that everyone can find and develop existing code, as well as add their own. Code repositories address this need and you'll be hard-pressed to find a developer in today's industry who doesn't use Git.

All these developments along the open-source timeline lead to a growing community of enthusiasts who have created numerous innovations together in software development. Container orchestration is one such example of how open-source software development has made it easier for programmers to access new ideas and collaborate with others to continuously improve their work and productivity.

Consider Docker, a more recent edition to the open-source landscape and a hugely successful container deployment program. This technology allows self-contained packages of software to be moved across different environments to help avoid computing conflicts and hasten deployment. Docker is also one of the most successful open-source tools; its adoption rate at the time of its launch was more significant than Git and cloud technologies. However, some might argue it is currently having difficulty stabilizing its business and revenue stream due to competition from other open-source tools like Kubernetes. Today, container orchestration is developing at a fast pace as many platforms, such as AWS, are rapidly providing offerings of their own to compete in the market.

The open-source movement has continually become stronger throughout history, having a significant effect on technology. But with this knowledge being "freely" available, how has it changed business?

The basic framework of a revenue-generating business model is to innovate, create, and market a product. Competition is derived from having a secret ingredient that makes a product special. Successful brands charge for exclusivity and the edge their products have over their rivals. Apple, with its unique product range and OS, is the perfect example of how "rarity" can drive profit.

Apple's antithesis is Android. Unlike Apple, Android is an open playing field for all, making great use of the open-source business model. It offers free open-source tools that its community can have a direct input on. So, how does anything open source make money? Open-source projects typically make a profit from upgrade packages that include support, private features, and usage on a larger scale such as "enterprise" packages for companies or large IT teams.

Both models work successfully in tandem because they reach and deliver to very different demographics. Many people wish to pay for the latest exclusive product upgrade, but there are just as many out there who will support the wealth of open-source projects now available on the market.

So, why should you consider open-source tools?

- **Security**: Open source revolves around transparency and inspection by a wide community invested in project success.
- **Quality code**: Due to the nature of its collaborative approach — and lack of boundaries — open-sources projects inherently generate high-quality work.
- **Freedom**: The freedom to adapt, collaborate, and develop with like-minded people. Open source is a step towards the future of free software for all. Get involved.

STEFAN THORPE is an IT professional with 20+ years of management and hands-on experience providing technical solutions to support strategic business objectives.

Tips for Launching Your First Open-Source Project

QUICK VIEW

01. This history covers the rise of Linux, Apache, and the wealth of open-source projects available today.

02. We look at the impact the open-source philosophy has had on software development.

03. We navigate through Git and what code repositories help teams achieve.

04. And finally, we examine the revolutionary business models that open-sourcing has generated, as well as their drive for a different ethos in programming.

BY CYNTHIA RICH
MANAGER OF TECHNICAL TRAINING, GITHUB

The year was 2005. The Linux kernel was already massive and it was in need of a new home. The community of committed developers that had built up around the kernel had outgrown its previous home and needed a new, better way to collaborate and maintain safe, reliable version control. In about ten days, this tight-knit community created the source code management we now know as Git.

Open source wasn't new in 2005. But it was about to take off in a way no one expected. The open-source world exists to bring together communities of developers bound by a shared goal. That goal is to make the world a better place with accessible, reliable software.

Your first project may not be the size of Git, but studying successful open-source projects like Git can give us important insights into why some projects succeed while others are unknown.

TIP 1: KNOW THE RECIPE

Everything great starts with a basic recipe – and so do great open-source projects. Here is the recipe for success in open source: solve real problems, make it easy to use, and find a community of passionate users.

SOLVE REAL PROBLEMS

One of my favorite open-source projects is **Probot**. Probot is a relatively new project. Its quick rise in popularity can be attributed to the fact that it fulfills a very real need for teams who are interested in automating certain aspects of their work. Probot provides an easy way to perform custom actions inside GitHub issues and pull requests.

It is essentially a webhook event handler. Webhooks aren't new – they have been around for years. What makes Probot unique is that it eliminates the need to build custom authentication processes and provides an easy API for interpreting webhooks. If I had a dollar for every time someone scratch-built these same processes over the last ten years... well, that's a topic for another day. The point is that Probot eliminates the pain points developers typically experience when setting up these types of services.

EASY TO USE

If your first goal is to solve for pain points, then your second goal needs to be ease of use. As a good rule of thumb, developers will only invest one hour in learning how to use your project – and that number is getting smaller every year. Great open-source projects have a predictable set of tools geared toward new users.

Electron is a great example of an open-source project that focuses on ease of use. The Electron project has robust documentation for getting started as well as a quick-start repository that you can clone and explore as a reference implementation. The community has contributed numerous sample apps that are designed to stretch your imagination even further.

PASSIONATE USERS

Successful open-source projects first need to be successful software projects. The relationship between an open-source project and its community is one of mutual fulfillment. Passionate users will have a vested interest in adding new capabilities and in making your project more secure and reliable.

The users of your software will become contributors to your project. How you treat these users will ultimately determine your success in building a community. In the beginning, you will play the role of product manager, senior developer, and community manager. As the community grows, you will find passionate contributors who share your vision for the project and are eager to support you by taking on some of these roles.

By creating a welcoming environment for your future contributors, coaching them, and accepting their changes, you show them that they are valued. These heavily invested users will share your software with their colleagues. This results in a cycle of value that repeats throughout the life of the project. To put it simply, if your project doesn't have any users, it will have very few, if any, contributors.

TIP 2: EMBRACE THE ROLE OF THE MAINTAINER

When you open-source your project, you put on a new hat: that of a maintainer. Gone are the days when you can spend all your free time gleefully adding new features to your project. Open-source projects are like gardens, they need constant tending and weed control.

TENDING THE PROJECT

When developers are contemplating whether to use an open-source project, they look for a few key things. These things include a License, a Code of Conduct, and some key indicators of project health:

- Is your project being actively maintained and updated? The world of software moves at a startling pace. If the last commit on your project was over six months ago, it may not be receiving critical updates.
- Are bugs being handled? Check the issues. If bug reports go unacknowledged or unresolved, this is a red flag to many prospective users and contributors.

As a maintainer, your role is to constantly tend to the project. Responding to issues and pull requests, coaching new contributors, and squashing bugs quickly will take time away from coding, but this a necessary trade-off for good project maintenance.

WEED CONTROL

Open-source communities are largely self-policing. As a maintainer, you set the tone for the community and ensure community guidelines are being followed. If a bad actor takes aim at your community, you will want to have tools in place to correct their behavior. One of your greatest assets in these scenarios is your Code of Conduct. The Code of Conduct signals that you will not accept harmful behavior and explains your process for dealing with it.

TIP 3: BE MINDFUL OF THE MIGRATION

Whether your project is currently on GitHub or your local machine, moving to GitHub means moving to a public repository. Before you make the move, consider what you may be unintentionally sharing with the world.

IS THE PROJECT IN A REUSABLE STATE?

You may need to make the project more dynamic before you make the move. For example, have you hardcoded values that others may need or want to customize? Can the project be generalized to other common scenarios without making changes to the code?

WHAT IS IN YOUR HISTORY?

If your project has been under version control, consider whether all of the history should be made public. For example, is the email address associated with your commits safe for sharing publicly? Have you ever accidentally committed a private key or token? If you aren't 100% sure whether your commits are safe to share, it may be time to start over with version control.

WHAT ABOUT ISSUES AND PULL REQUESTS?

Issues and pull requests created when the project was private contain important historical information, but they may also contain information you never intended to share publicly. It may be wise to move the project to a fresh repository when you make the switch from private to public.

SHOULD YOU MAINTAIN A MIRROR PROJECT?

If you will need to maintain some level of differentiation between the project you use internally and the one you are moving to open source, consider how you will pull updates in the open-source project back into your internal project. This is best handled by pulling the open source project in as a dependency.

TIP 4: SETTING UP FOR SUCCESS

Before you make your project public, you should also spend time thinking about the contributor's experience when they visit your repository for the first time. The open-source community has adopted a set of documents to help make this experience more consistent between projects. If you haven't done it already, you should add these at the root level of your project.

LICENSE

Without a license, it isn't open source. Every open-source project must have a license. The license you choose is important because it dictates how your project can be used, attribution requirements, and contribution of changes. Changing or adding a license after making your project public can have some serious implications, so make license selection your first step.

For a full discussion on licensing, check out choosealicense.com. An interactive guide will help you choose the perfect license and you can even copy and paste an existing license into your project. You should not modify the language in the license, aside from replacing any placeholder text.

README

Think of the README as a welcoming entryway into your project. It should include information such as:

- What the project does
- How it is commonly used
- How to get started with the project
- Who maintains the project
- How to get started contributing
- The goals of the project

CONTRIBUTING

Contributing to a project takes the relationship to a whole new level. Most maintainers opt to provide contribution details in a separate document. This document should be called the CONTRIBUTING guide and is highlighted for users who are opening issues and pull requests in the repository.

Great contributing guides provide the following details:

- The types of contributions you accept
- The preferred process for proposing and contributing changes
- The process of building and testing
- An overview of the system architecture and dependencies
- The location of most commonly updated files

CODE OF CONDUCT

Healthy communities conduct themselves in a kind and inclusive manner. To ensure your project's community is healthy, adopt a code of conduct. The code of conduct provides guidelines for the behavior you expect in the project. There are several great drop-in options you can use such as the contributor covenant found at contributor-covenant.org.

TIP 5: BE PREPARED TO FIND AND NURTURE YOUR COMMUNITY

We mentioned earlier that many open-source projects never gain a community simply because they are unknown. Let's discuss specific strategies to prevent your project from becoming one of the unknowns.

HAVE A PROJECT LANDING PAGE

Don't depend on the GitHub repository as the only face of your project. Your project should have a landing page that helps users who are searching for a solution find the one you have created. Be sure to link back to your GitHub repository in the footer to make it easier for your future contributors to find.

TALK ABOUT YOUR PROJECT

You don't have to love public speaking to promote your project, but it doesn't hurt. If the thought of speaking at conferences sounds terrifying, try writing about it instead. Consider creating a blog where you can tell users about new features and give users the opportunity to share their stories. You can also record videos and create tutorials.

OPENLY DISCUSS FRICTION

Some projects create a repository within the GitHub organization dedicated to discussing friction points in using and contributing to the project. Others prefer issues in the main repository. Regardless of where the discussions take place, your job as a maintainer is to be open and welcoming of these discussions. If you respond with, "You're doing it wrong," or, "We're not going to change that," you risk losing a user and a potential contributor.

Great documentation helps ensure the friction experienced isn't user error – and can become a good way for these users to make their first contribution by clarifying the documentation for others.

PROVIDE MEANINGFUL CONNECTIONS

Discussions in issues and pull requests are great, but they don't help people form real bonds. Consider adding methods for face-to-face connections. For example, the Probot team holds regular office hours via video conference. Larger projects like Git have in-person gatherings and conferences.

PROMOTE CONTRIBUTORS TO MAINTAINERS

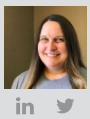
Ultimately, this whole business of maintaining an open-source project can be lonely work. Identifying and promoting active contributors allows you to share the work and allows them to contribute in a more meaningful way. You can start by asking them to help you triage issues and pull requests, then consider allowing them to review and approve certain types of pull requests using the CODEOWNERS file. Ultimately, you will find a small number of dedicated individuals you trust to merge pull requests and even help define the strategic direction of the project.

TAKING THE NEXT STEP

Maintaining an open-source project is challenging but rewarding work. Follow these five tips and you will be well on your way to success.

What will you open source?

CYNTHIA RICH is the Manager of Technical Training at GitHub. When she isn't in the classroom teaching teams about collaborative development, Cynthia is passionate about helping developers and organizations succeed in open-source. You can find Cynthia on GitHub at @crichID or on Twitter at @cynthiarich07.



TAKE **CONTROL** of Your **Open Source Software**

KNOW WHAT'S IN YOUR CODE

MANAGE
Open Source Security and Compliance

COMPLY
with OSS License Obligations

IDENTIFY
Open Source Security Vulnerabilities

INTEGRATE
Scanning into Your DevOps Environment

Never Miss Evidence of Open Source Software - From Software Packages to Code Snippets.
FlexNet Code Insight - an end-to-end software composition analysis platform.

Flexera

1.800.809.5659 | +44 870-871-1111 | **FLEXERA.COM**

Managing Risk and Compliance with Software Composition Analysis

Technology companies now depend on open-source for up to 90% of the software that they build into their tech stack and products. This change happened so quickly that most companies have been left exposed to the vulnerability and compliance problems that come from not tracking and managing their open-source components. Additionally, the scale of open-source use has overwhelmed the traditional processes for managing use of third-party software.

Today, solutions exist that are designed to help organizations keep track of their use of open-source and commercial dependencies, to understand their open-source license obligations, and receive alerts when new vulnerabilities are discovered. This allows them to develop products that do not contain known vulnerabilities when shipped and to stay on top of components as they age-out when deployed. This type of scanning and software management is known as Software Composition Analysis (SCA).

SCA software contains scanning and workflow features designed to help technology companies discover, manage, upgrade and comply with their use of open-source components. By scanning and comparing the files used on the devices to a database of billions of known open-source files, the system can discover usage of third-party components for the purposes of vulnerability management as well as open-source license compliance. This enables organizations to create their Software Bill of Materials down to the snippet level—a growing customer contract requirement. Managing these requirements allows an organization to ship a product that respects the open-source community, as well as protect the company's users from attacks.

WRITTEN BY JEFF LUSZCZ
VICE PRESIDENT OF PRODUCT MANAGEMENT, FLEXERA

PARTNER SPOTLIGHT

FlexNet Code Insight

Application and device security, open-source scanning, and software vulnerability management at all stages of the product lifecycle.

CATEGORY
Software Composition Analysis

OPEN-SOURCE?
No

NEW RELEASE
Quarterly

CASE STUDY

Flexera customers use FlexNet Code Insight to reduce software vulnerability risk and manage license compliance for open source software (OSS) and third-party components. FlexNet Code Insight scans application source code and, if necessary, issues alerts if vulnerabilities are identified. Customers rely on Flexera for:

- **Open Source Package Analysis** – scan code early and often during the development process to identify what you are using and build a Bill of Materials (BOM) for your products and take control of your open source software management.
- **Flexible Scans** - adjust the depth and breadth of scans and analysis based on the scope and the risk of your projects. Find all evidence of open source in binaries, containers, build dependencies, subcomponents, modified and partial open source components.
- **M&A and Baseline Audits** - Highest standard of security and confidentiality for mergers and acquisitions, and objective third-party baseline audits with recommendations for remediation.

STRENGTHS

- Patented scan and analysis flexibility
- Largest open-source knowledge base of over 14 million components
- Proactive and continuous monitoring
- Flexible scans based on risk profiles
- Integrates into the build cycle

WEBSITE www.flexera.com/sca **TWITTER** @flexera **BLOG** blogs.flexera.com/sca

Addressing the Complexity of Big Data With Open Source

BY KONSTANTIN BOUDNIK
MEMBER OF EPAM'S OPEN-SOURCE ASSOCIATE SCHOLAR NETWORK

QUICK VIEW

01. The Big Data stack, created from more than 20 different projects developed by committers and contributors of Apache Software Foundation, is very complicated.

02. Apache Bigtop™ provides and enforces the best practices and operations of software development to the lifecycle of any complex stack.

03. Apache Bigtop enables users to reduce the cost of deployment and production by creating the packages from known building blocks to a different environment so they will work the same way no matter where they are deployed.

Simple software is a thing of the past. Think about it: No program out there is created in a vacuum. Every program uses libraries, has run-time dependencies, interacts with operational environments, and reacts to human inputs. Free and open-source software, as a creative free market approach to software development, provide more than one solution for every challenge. There are multiple compilers, operating systems, statistics packages (known today as machine learning), test frameworks, orchestration solutions, and so on. Each project moves at its own speed, releasing new features and adding new attributes. Imagine for a second that there is a need to combine a few of these complicated projects into a meta-complex system. It sounds quite sophisticated, doesn't it?

THE BIG DATA ZOO

Just like a zoo with hundreds of different species and exhibits, the big data stack is created from more than 20 different projects developed by committers and contributors of the Apache Software Foundation. Each project has its own complex dependencies structure, which, in turn, build on one another very much like the Russian stacking doll (matryoshka). Further, all of these projects have their own release trains where different forks might include different features or use different versions of the same library. When combined, there are a lot of incompatibilities, and many of the components rely on each other to work properly, such as the case of a software stack. For example, Apache HBase and Apache Hive depend on Apache Hadoop's HDFS. In this environment, is it even possible to consistently produce software that would work when deployed to a hundred computers in a data center?

REINING IN COMPLEXITY

A software stack is an ultimate bundle that can be given to a customer as a package or a container image and is defined by a bill of materials specifying the exact names, versions, and patch level for each component. Customers expect that each individual software project has been carefully selected and altered so that it works with the rest of the distribution, and that everything has been validated at the integration and system level.

> Controllability has the power to lower the cost of development. That's why we use version control, the CI/CD process, and automated deployments.

Fortunately, this issue has already been resolved in operating systems. Debian Linux is probably the most famous, as it has allowed for the creation of many derivatives and offspring over the years. By understanding the principles required to create complex software systems like an operating system, is it possible to build a solution that creates a complex stack based on Java, C, or scripting languages, or a combination of them? This was the question that a number of experts attempted to

address in producing and deploying Hadoop's stacks in the early days of the platform.

Controllability has the power to lower the cost of development. That's why we use version control, the CI/CD process, and automated deployments. Apache Bigtop was conceived as a framework that would provide and enforce the best practices and operations of software development to the lifecycle of any complex stack. Bigtop graduated to an ASF top-level project in 2012. At a high level, it consists of the following pieces: packaging code, deployment code, a Docker-based mechanism to create clusters for development and testing, distributed integration test framework, tests for stack validation, and a build system written in Gradle (a DSL of Apache Groovy). The development lifecycle looks like this: write and commit the code (either for Bigtop itself or for one of the components referenced by its bill of materials), run the build, fire up a cluster (either Dockerized or real if you have spare hardware handy) using the provided Puppet recipes for deployment and configuration of the fresh packages, run the tests, and analyze the results. Repeat as many times as needed.

All of these moving parts effectively serve one purpose: to create the packages from known building blocks and transfer them a different environment (dev, QA, staging, and production) so that no matter where they are deployed, they will work the same way.

Does this sound familiar? You bet! This is pretty much how any well-constructed software pipeline is done today. There is one key difference, though: Bigtop users are dealing with highly complex systems that require some serious orchestration. It isn't exactly your grandpa's mobile app. Yet, Bigtop makes the process of a software stack creation so seamless that I was able to produce a fully equipped and tested commercial distribution including Hadoop, HBase, Hive, and more in under four weeks with the help of a small team of engineers.

STREAMLINING OPERATIONS

All of these moving parts effectively serve one purpose: to create the packages from known building blocks and transfer them a different environment (dev, QA, staging, and production) so that no matter where they are deployed, they will work the same way. The deployment mechanism needs to control the state of the target system. In other words, this means that if you use the same source code and run the same build and deployment over and over again, you will end up with the same result every time.

Relying on a state machine like Puppet or Chef has many benefits. You can forget about messy shell or Python scripts to copy files, create symlinks, and set permissions. Instead, you define "the state" that you want the target system to be, and the state machine will execute the recipe and guarantee that the end state will be as you specified. The state machine controls the environment instead of assuming one. These properties are great for operations at scale, DevOps, developers, testers, and users, as they know what to expect.

BUILDING THE ECOSYSTEM

Now you know why Bigtop is the framework used by all vendors of the Hadoop ecosystem. Amazon EMR, Google Dataproc and others are using it extensively. **ODPi** has chosen Bigtop as its foundation of the industry's first reference architecture for a Hadoop-based stack.

It's hard to overestimate the importance of open standards in the model software industry. If you're an application developer, your costs would increase significantly if you had to deliver your product to two different platforms. Wouldn't it be cool if you could certify against a reference architecture and then your code would work on all compatible clusters? That's exactly what Bigtop will help you achieve while also reducing the cost of development and production.

KONSTANTIN BOUDNIK is one of the veteran developers of Apache Hadoop and co-author of Apache BigTop, the open-source framework for creation of software stacks and operation of data processing projects used by all commercial vendors of Hadoop-based platforms including Amazon EMR, Google Cloud Dataproc and other major distributors. Dr. Boudnik has more than 20 years of experience in software development, big- and fast-data analytic, distributed systems and more.

OPEN SOURCERY

Everyone wants to take advantage of the magic of open-source software, whether it's to develop new tools, improve products, or just for your own personal satisfaction. However, if you want your project to grow and solve more problems, you need to solve the equation of attracting people to your project. Without a strong community to welcome newcomers, create best practices, and fix new bugs, there's little benefit to allowing everyone to have access to your code. To brew your perfect open-source solution, you can't use any magic words, but you'll need four key ingredients.

CONTRIBUTORS

The foundation of a good community is what people contribute. Without contributions, there is no benefit to using open source — it's free software from one person. What's especially problematic is 88% of survey respondents use OSS in their jobs, but only 16% contribute back!

DOCUMENTATION

To get new users started with a project, they need to be able to see the code and have easy-to-use documentation. Of contributing respondents, 34% write and 37% improve documentation. At the same time, 67% of respondents have issues with unclear or unavailable documentation, so OSS communities have a lot of work to do.

IMPROVEMENTS

When most people think of OSS contributors, they think of people who improve the code. And of contributing respondents, 54% focus on adding new features, and 68% focus on fixing bugs. This is the not-so-secret sauce to the appeal of OSS.

MATURITY

Maturity refers not just to the age of the solution, but what processes are in place to contribute to a solution and how changes are implemented. These kinds of documents are usually based on the Client Maturity Model. 63% of respondents say maturity helps them pick a particular open-source solution.

DIVING DEEPER

INTO OPEN-SOURCE

#OPENSOURCE TWITTER ACCOUNTS

 **@enunomaduro**

 **@benbalter**

 **@jllord**

 **@rikkiends**

 **@cra**

 **@ibrahimatlinux**

 **@jeresig**

 **@JessRudder**

 **@sophshepherd**

 **@lolamby**

OPEN-SOURCE ZONES

Agile

dzone.com/agile

In the software development world, Agile methodology has overthrown older styles of workflow in almost every sector. Although there are a wide variety of interpretations and techniques, the core principles of the Agile Manifesto can help any organization in any industry improve their productivity and success.

DevOps

dzone.com/devops

DevOps is a cultural movement supported by exciting new tools that is aimed at encouraging close cooperation within cross-disciplinary teams of developers, and IT operations, and system admins. The DevOps Zone is your hot spot for news and resources about Continuous Delivery, Puppet, Chef, Jenkins, and more.

Java

dzone.com/java

The largest, most active Java developer community on the web. With news and tutorials on Java tools, performance tricks, and new standards and strategies that keep your skills razor-sharp.

OPEN-SOURCE REFCARDZ

Getting Started With Git

This updated Refcard explains why so many developers are migrating to this exciting platform. Learn about creating a new Git repository, cloning existing projects, the remote workflow, and more to pave the way for limitless content version control.

Getting Started With OpenStack

Introduces OpenStack, an open-source IaaS platform used to manage large pools of compute, storage, and networking resources in a data center.

Getting Started With Kotlin

Kotlin has become one of the most popular JVM languages in the past few months, partly because it experienced a lot of attention in the Android community after Google made Kotlin an official language for Android development.

OPEN-SOURCE PODCASTS

Floss Weekly

Learn about the most interesting and important news and opinions in the open-source community from a rotating panel of experts.

GNU World Order

Get in-depth tutorials on a wide range of open-source software tools and workflows.

The Linux Action Show

Learn about current news that pertains to Linux and open-source topics, as well as the hosts' experiences using various open-source tools.

OPEN-SOURCE RESOURCES

Open-source Resources

Browse a collection of resources to learn how to get started in open-source, learn about the different facets of open-source, and more.

Trending on GitHub

Check out a real-time log of the trending open-source projects on GitHub.

The Essential Open-source Reading List

Get expert advice on how to get your open-source tool collection started, how to appraoch issues such as licensing and governance, and more.

Community Standards in FOSS

BY LISA SMITH
FRONT-END DEVELOPER, **SPOONFLOWER**, AND ZONE LEADER, **DZONE**

QUICK VIEW

01. Codes of conduct need to be explicitly and well-enforced.

02. Trying to avoid implementing a code of conduct to not drive away busy contributors will hurt your project in the long run.

03. Instead of a list of "don'ts," try a list of "do's," like "be welcoming."

One of open-source software's greatest assets — the egalitarian structure of group ownership — introduces one of its thorniest problems— enforceable community standards. As participation in open-source projects grows, so do the problems inherent in any group setting, namely how is problem behavior addressed, corrected, and if needed, punished. Balancing protection and participation is a struggle experienced by projects large and small alike.

Throughout 2017, tech companies reckoned with their bias, harassment, and abuse issues in very public ways. Most of these companies, however, had HR departments or reporting structures in place, though in many cases they were ineffective. Open-source projects lack both of these, which both exacerbates the problem and allows it to go un- or under-reported. Frequently, targets of abuse are forced out of communities either directly through harassment or subtlety by lack of support. Compounding this lack of support is the guilt that goes along with being perceived as unsupportive of the community when reporting problems or ultimately leaving projects. Problematic projects can be toxic, support collaboration in name but not in spirit, make newcomers feel unwelcome, espouse a culture that is exclusionary in word or deed, and lack clear pathways to leadership for interested parties who are not already in such roles.

Technology in general suffers from a lack of diversity and struggles with inclusion and the communities surrounding open-source projects are frequently even more difficult for underrepresented groups to penetrate because of their structure. It takes intention and work to make a community inclusive and if no one in leadership is attentive to addressing these issues it is easy to overlook or ignore. Groups working with AI, for example, will fail to address vital components in their algorithms by not addressing inclusivity in their development process. As with the larger tech community, inclusion and diversity are not simply pipeline problems but systemic hurdles that need to be overcome with support, mentoring, and active interventions.

Community standards and codes of conduct in open-source communities are important tools to strengthen inclusion, encourage retention, and expand opportunities for all members to participate at every level. Systems need to be put in place to not only encourage contribution from all kinds of participants, but to address flawed interactions and abuses of power and privilege.

Many communities have relied on a "Be excellent to each other" or "Just be nice" policy that sounds kind and hip and breezy, but lacks structure, enforcement, concrete examples of what constitutes unacceptable behavior and consequences. Long-standing members cannot be perceived as untouchable simply by virtue of tenure or number of contributions. Allowing misconduct to stand from any member is unacceptable, but when high-volume contributors are given a pass because they are "valuable" to a project, that only serves to make everyone else feel undervalued

or worthless entirely. By trying to minimize the risk of "offending" some users, many are effectively silenced. Open-source communities frequently suffer from the fallacy of composition, or believing that what is good for one member is good for all members. Because open-source projects are at their foundation, many people coming together around a shared goal their intentions are always believed to be good despite examples of misbehavior.

Other concrete steps include understanding implicit biases and how to adapt and overcome them, working to use inclusive language at all times, not just when it appears convenient, calling out misbehavior when it occurs and addressing the same in as positive a manner as possible without invalidating the feelings of those affected.

Limited resources are frequently cited as the reason for not investing in governance, but that is a fallacy of short-term economics that fails to take into account future productivity losses when good potential contributors are driven away from toxic projects. Investing in a sustainable model where positive outcomes are encouraged, like "be welcoming," instead of trying to be flippant by asserting "don't be a jerk" leads to more sustainable inclusion policies. Quality and kindness do not have to be mutually exclusive. And this investment doesn't have to be costly - as a leader in a large open-source project wisely said, "Don't roll your own Code of Conduct." This hard work has been done already. The issue that is more important is to actually adopt a code of conduct and enforce it. Communities that are reluctant to adopt CoC's for fear of alienating core contributors do so to their own detriment. Having a CoC that is watered down or isn't enforced is as bad or worse than having none at all. Paying lip service to the concept helps no one. In a 2017 survey on diversity and inclusion in open-source, Mozilla found that the majority of respondents thought that even if their community had a CoC they were skeptical that it was enforced at any level.

To combat this skepticism and inertia, Mozilla has proposed a scalable system for community guidelines that addresses all of the waypoints between discovery and resolution. Discovery includes accessible information on how to identify and report issues and timelines for the full process as well as specifying who has access to that information since fear of retribution or exposure is a powerful deterrent in reporting.

After discovery comes reporting — a form-based system of easily documenting and quantifying incidents. Once reporting has happened the triage step tags incidents tying them to action triggers and additional communication.

Once an incident has been reported and triaged, it must be investigated and acted on by the appropriate groups, identified by roles defined in a RASCI — who is Responsible, who will be held Accountable, who is in a Supportive role, who should be Consulted, and who is Involved.

Following investigation, a recommendation must be made, based on an escalation of response to severity of violation. They have proposed a 5-level escalation ladder from no action to revocation of access to tools and communities.

Once a recommendation has been made, notification of this recommendation must be made to the reporter and the reported. Follow-up will then include concrete plans of action, including the safety of people in situations of risk.

The final step of resolution is only achieved when all the actions of the recommendation have been completed and all parties have been notified.

In the course of proposing this system, the team working on the project reports learning many valuable lessons including the importance of providing and ensuring anonymity for reporters, making sure that resolutions include both teaching and learning, not just punitive actions, making sure that reinstatement procedures include intensive on boarding to try to set up future successes, and ensuring that self-care is part of the process for the front-line members of the community handling these issues. You can read all about Mozilla's process here. and contribute to their Diversity and Inclusion repo on Github.

2018 has been pledged as the year open-source improves itself by making participation safe and satisfying for participants at every level. Organizations like Stack Overflow are admitting their shortcomings and making concrete plans to address them. Open-source needs to grow and adapt as a whole so that projects can continue to flourish and a strong community of participants grows far into the future.

LISA SMITH is a former librarian turned web developer. She's been working online since before there was a Google. Lisa works at Spoonflower where she does front-end development. In addition to writing clean, usable code, Lisa cooks and bakes for friends and family, often testing untried recipes on large unsuspecting crowds. She likes Guitar Hero, impromptu living room raves, and spending time with her two daughters.

Executive Insights on the Current and Future State of Open-source Software

QUICK VIEW

01. Companies are frequently using open-source to analyze Big Data whether it be streaming or for AI-powered digital automation.

02. Companies are also using open-source for databases, automation, SaaS, storage, platforms, and testing.

03. The most frequently mentioned open-source software are Linux, Kubernetes, Docker, and several projects from both Eclipse and Apache.

BY TOM SMITH
RESEARCH ANALYST AT **DZONE**

To gather insights on the current and future state of open-source software (OSS), we talked to 30 executives. This is nearly double the number we speak to for a research guide and believe this reiterates the popularity of, acceptance of, and demand for OSS. Here's who we spoke with:

- **Anthony Calamito**, Chief Geospatial Officer, **Boundless**
- **Jakob Freund**, CEO of **Camunda**
- **Pete Chestna**, Director of Developer Engagement, **CA Veracode**
- **Julian Dunn**, Director of Product Marketing, **Chef**
- **Matt Ingenthron**, Senior Director of SDK Engineering, **Couchbase**
- **Stephan Ewen**, Co-founder & CTO, **Data Artisans**
- **Amol Kekre**, Co-founder and Field CTO, **DataTorrent**
- **OJ Ngo**, Co-founder & CTO, **DH2i**
- **Stefano Maffulli**, Director of Community, **DreamHost**
- **Kelly Stirman**, CMO and VP Strategy, **Dremio**
- **Konstantin Boudnik,** CTO Big Data & Open-source Fellow, **EPAM**
- **Tyler McMullen**, CTO, **Fastly**
- **Jeff Luszsz**, VP of Product Management, **Flexera**
- **Angel Diaz**, V.P. Developer Technology & Advocacy, **IBM**
- **Ben Slater**, Chief Product Officer, **Instaclustr**
- **Grant Ingersoll**, CTO, **Lucidworks**
- **C J Silverio**, CTO, **npm**
- **Mark Gamble**, Sr. Director of Product Marketing, Analytics, **OpenText**
- **Francis Dinha**, CEO, **OpenVPN**
- **Sirish Raghuram**, CEO & Co-founder, **Platform9**
- **Neil Cresswell**, Co-Founder, **Portainer.io**
- **Lars Knoll**, CTO, **Qt**
- **Brad Adelberg**, Vice President of Engineering, **Sauce Labs**
- **Giorgio Regni**, CTO, **Scality**
- **Dor Laor**, CEO, **ScyllaDB**
- **Harsh Upreti**, Product Marketing Manager, API Products, **SmartBear**
- **Jean-Baptiste Onofre**, Technical Fellow & Software Architect, **Talend**
- **Antony Edwards**, CTO, **Eggplant**
- **Matt Ellis**, Architect, **TIBCO Software**
- **Karthik Ranganathan**, Co-founder & CTO, **YugaByte**

KEY FINDINGS

The response to our query for executive insights on the current and future state of open-source was overwhelming, with nearly twice the number of responses we have received for other research guide topics. I believe this speaks to the popularity of the open-source movement, as well as the fact that this is DZone's first research guide covering the open-source ecosystem.

1. The companies from whom we received feedback are most frequently using open-source software to **analyze big data, whether it be streaming or for AI-powered digital automation**. It is also frequently used for databases, automation, SaaS, storage, development platforms, and testing.

2. The most popular open-source foundation is **Apache, with**

18 different projects mentioned. Linux was mentioned second most frequently, followed by Kubernetes, Docker, and projects from the Eclipse Foundation. Speaking to the breadth of the open-source ecosystem is that more than 75 languages were mentioned by respondents with one respondent mentioning they use around 50 different libraries in their solution.

3. The most important elements of the open-source ecosystem are an active **community of users and contributors** who share their knowledge, experience, and insights to improve collaboration, innovation, code quality, and security of the code. Existing users care about getting bugs fixed and enhancing the code, and there is also a good balance of people pushing the boundaries with new ideas and features. The community fosters innovation by being transparent, having a culture of meritocracy, and being able to communicate directly with the people who created the software.

It's impossible for the original creators of a project to design perfect software that meets the needs of a diverse set of users. That's why collaboration is so important and why open-source solutions evolve so quickly – the power of community.

4. The most important players in the open-source ecosystem are **The Apache Foundation, The Linux Foundation, Red Hat, and big tech companies that are developing a lot of open-source components** for themselves but are sharing with the community. Apache is mentioned most frequently due to the incredible number of libraries and projects. The Linux Foundation is recognized for their licensing model. Big tech companies mentioned include: Amazon, Apple, Google, IBM, Microsoft, Netflix, and Twitter.

5. The most significant changes to the open-source ecosystem is the **growth of the ecosystem and the entry and adoption by large companies**. According to Sonatype, there are more than 10,000 new open-source versions per day and big-name vendors such as IBM, Microsoft, Oracle, and VMWare are beginning to release more and more of their products under open-source licenses.

The ecosystem has been adopted by more software companies and has identified a viable business model that is very attractive to the investment community. Everything is accelerating – the number of projects, the speed at which products leapfrog their legacy counterparts, as well as the number of developers participating. When large companies donate already-complete projects to the community, they take off like wildfire because they're already usable having been incubated inside those companies for years.

6. Nearly all real-world problems are being solved by open-source software, since 90 to 95% of all apps are being built with at least some open-source components. These technologies are solving the problems that proprietary software had been solving previously – messaging, databases, microservice frameworks, etc. The software is allowing for innovative solutions to be brought to solve problems with cybersecurity, private access, scaling database, scaling cloud infrastructure, software management and provisioning, and more. Because the vast community that supports open-source is constantly innovating, OSS allows users to respond faster to change, which enables IT to do more and businesses to be more responsive and data-driven.

A better question would have been, "What are the real-world problems not being solved by open-source software today?"

7. The most common problem with the open-source ecosystem are: **1) the accrual of technical debt; 2) complexity; and, 3) license issues**. There is generally a failure to address technical debt that is implicit with open-source. You need to be aware of the technical debt that can accrue very quickly. You need to spend the time to update the code and tools and spend the time and effort necessary to build changes into the project. People forget how important and critical maintenance is and don't invest in it.

Use of open-source technologies without a hardened stack requires expertise that is very rare. Open-source projects are not necessarily built so they can be dropped into a production environment out of the box. Having an easy and lightweight way to handle continuous operations and support frictionless application development can be a big challenge for many teams. The speed of change and incredible diversity of projects is challenging.

A lot of organizations do not understand open-source and the licensing obligation that goes along with it. There are still legal matters around open-source that are not entirely solved. Many projects lack licenses, or lack compliance with their own license.

8. **The future of open-source is bright and is going to get brighter**. Open-source has won and will become the default model for most software projects. It will continue to be integrated in different ways to solve problems and achieve results in big data, AI/ML, blockchain, IoT, and wearables. The

proliferation of frameworks will continue. Open-source will play an even larger role as unique business logic is layered on top.

9. Concerns over the current state of the open-source ecosystem were far ranging, with those most frequently mentioned related to: **1) obsolescence; 2) overlap; 3) vendor mentality; 4) domination by large players; and 5) consumption without contribution**.

The biggest concern is the obsolescence of a huge chunk of the open-source ecosystem every year. Many of these software projects become obsolete because there are not enough people contributing to them. Many open-source projects do not have a large distributed community around them, and often depend on a single corporation to make most of their contributions and determine their roadmap. Once the corporation has gotten what they want from the code, they no longer support it.

There are too many overlapping products competing against each other. It can get overwhelming for the end user as there are so many choices with minor differences.

Too many users view open-source projects as if they are software vendors and expect they can just log a defect with the project and someone will have responsibility to fix it.

Having an open-source license to a huge codebase is not a guarantee of long-term sustainability of the project if most of the knowledge is owned by one corporation. The consolidation of IT giants is a big issue that needs to be watched carefully. Few companies may end up owning the source code of too many major projects. Distributed copyright ownership would help mitigate risk.

Vendors also consume more than they share. There's a tendency for companies to use open-source systems, but not contribute to them. Community and users can choose open-source vendors that provide more open code and make contributions to support the ecosystem. The open-source ecosystem can only survive if people give back. If maintenance and development is ignored, software systems will falter and fade.

10. When asked, "How do you ensure the security of open-source software?" the most frequent response I received was that open-source software is inherently more secure than other code due to the number of people looking at it with the ability to discover vulnerabilities and fix them. The most answers to the questions revolved around: **1) follow best practices; 2) test; and 3) choose the most popular code**.

Adopt strict coding standards and guidelines. Ensure the software you are using complies with information assurance policy standards. Have an automated bill of materials and upgrade versions of open software when new versions come out. Perform manual auditing via code review and automate CVE database monitoring.

Perform vulnerability scans and publish the results every time the code is committed. Have a security vetting and testing process for each project that is out in the open so users can judge whether this is sufficient for their needs. Research, test, and read the license carefully. Always test security and audit for dependencies.

Make sure you are using the most popular code since it is less likely to have undiscovered vulnerabilities.

11. The two things developers need to keep in mind with open-source software are: **1) knowing, and being a good steward of, the unwritten rules of the community; and 2) learning and paying attention to the details of the licenses**.

Be comfortable interacting with the community. Consider the strength of the community before adopting an open-source product. Embrace open-source and start to contribute to open-source projects that are of interest to you. Don't be afraid to speak up and communicate. If you find a bug, report it. If you're able to fix the bug yourself, do it and help the community. Engage with the community on IRC and Slack, many of them are very receptive to newbies. Keep in mind that you are coding on the back of thousands of developers that came before you and it's important to leave the community and the project better than you found it. Encourage your company to dedicate time and money to supporting open-source projects.

You also need to understand the legal implications of using open-source code: read the licensing text and understand the terms. Consider using a package with fewer features that are more open. Be aware there may be legal compliance issues that you have not even thought about. Respect the license the author selected, and if you are not able to do so, do not use the component.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

Solutions Directory

This directory of popular open-source tools and foundations provides comprehensive, factual comparisons of data gathered from third-party sources and the tool creators' organizations. Solutions in the directory are selected based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

COMPANY	PROJECT	TYPE	WEBSITE
Adobe	PhoneGap	Hybrid web development platform	phonegap.com
Apache Foundation	OpenNLP	Natural language processing toolkit	opennlp.apache.org
Apache Foundation	SparkMLlib	Scalable machine learning library for Apache Spark	spark.apache.org/mllib
Apache Foundation	Lucene	Search software	lucene.apache.org
Apache Foundation	Solr	Search server built with Lucene	lucene.apache.org/solr
Apache Foundation	Drill	"Distributed queries on multiple data stores and formats"	drill.apache.org
Apache Foundation	Flink	Streaming dataflow engine for Java	flink.apache.org
Apache Foundation	GraphX	Graph and collection processing on Spark	spark.apache.org/graphx
Apache Foundation	Hadoop	MapReduce implementation	hadoop.apache.org
Apache Foundation	HDFS	Distributed file system (Java-based, used by Hadoop)	hadoop.apache.org
Apache Foundation	Ignite	In-memory data fabric	ignite.apache.org
Apache Foundation	Kafka	Distributed pub-sub messaging	kafka.apache.org

COMPANY	PROJECT	TYPE	WEBSITE
Apache Foundation	Mahout	Machine learning and data mining on Hadoop	mahout.apache.org
Apache Foundation	Spark	General-purpose cluster computing framework	spark.apache.org
Apache Foundation	YARN	"Resource manager (distinguishes global and perapp resource management)"	hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
Apache Foundation	CloudStack	Virtual machines	cloudstack.apache.org
Apache Foundation	Cassandra	NoSQL key value database	cassandra.apache.org
Apache Foundation	HBase	Wide column database	hbase.apache.org
Apache Foundation	ActiveMQ	Message queue	activemq.apache.org
Apache Foundation	Camel	Integration framework	camel.apache.org
Apache Foundation	Tomcat	Servlet container and web server (JSP, EL, Websocket)	tomcat.apache.org
Apache Foundation	TomEE	Tomcat and Jakarta EE features (CDI, EJB, JPA, JSF, JSP, and more)	tomee.apache.org
Apache Foundation	Maven	Build automation	maven.apache.org
Apache Foundation	Zookeeper	Service discovery	zookeeper.apache.org
Apache Foundation	Cordova	Hybrid web development platform	cordova.apache.org
Appium	Appium	Mobile test automation	appium.io
Arduino	Arduino Product Line	Open source prototyping platform	arduino.cc
Azul Systems	Zulu	Enterprise-grade OpenJDK build	azul.com/products/zulu
Bootstrap	Bootstrap	Web framework	getbootstrap.com
Buoyant	Linkerd	Service mesh	linkerd.io
CA Technologies	Automic	Release automation	automic.com
Canonical	Ubuntu Core	IoT operating system	ubuntu.com/core

COMPANY	PROJECT	TYPE	WEBSITE
Canonical	LXD	Container management	linuxcontainers.org/lxd
Chef	Chef	Configuration management and continuous automation platform	chef.io
Cloud Foundry	Cloud Foundry	Platform-as-a-service	cloudfoundry.org
CloudBees	Jenkins	Continuous integration	jenkins-ci.org
CoffeeScript	CoffeeScript	Development language	coffeescript.org
Couchbase	Couchbase Server	NoSQL database	couchbase.com/products/server
Docker	Docker	Software containers	docker.com/community-edition
DocumentCloud	Backbone.js	JavaScript framework	backbonejs.org
Drupal	Drupal	CMS	drupal.org
Eclipse Foundation	Jakarta EE	Java libraries	oracle.com/technetwork/java/javaee/overview
Eclipse Foundation	MicroProfile	Java microservices optimization projects	microprofile.io
Eclipse Foundation	BIRT	Visualization and reporting library for Java	eclipse.org/birt
Eclipse Foundation	Kura	Connectivity middleware	eclipse.org/kura
Eclipse Foundation	Vorto	IoT platform	eclipse.org/vorto
Eclipse Foundation	Eclipse	IDE	eclipse.org
Elastic	ElasticSearch	Distributed search and analytics engine	elastic.co
Elastic	Kibana	Elastic stack configuration and management	elastic.co/products/kibana
EnThought	SciPy	"Scientific computing ecosystem (multidimensional arrays, interactive console, plotting, symbolic math, data analysis) for Python"	scipy.org
Facebook	Infer	Static analyzer	fbinfer.com/
Facebook	React.js	Web framework	facebook.github.io/react

COMPANY	PROJECT	TYPE	WEBSITE
FitNesse	FitNesse	Acceptance testing framework	docs.fitnesse.org/FrontPage
Flexera	FlexNet Code Insight	Commercial software composition analysis	flexera.com/products/software-composition-analysis/flexnet-code-insight.html
Git	Git	Software configuration management	git-scm.com
Google	AngularJS	JavaScript framework	angularjs.org
Gradle	Gradle	Release automation	gradle.org
Grails	Grails	Java web framework	grails.io
Grakn Labs	Grakn	Relational database for AI apps	grakn.ai
H2O.ai	H2O	Machine learning platform	h2o.ai/h2o
Hashicorp	Vagrant	Configuration management	vagrantup.com
Hashicorp	Consul	Service discovery, configuration, and monitoring	consul.io
Hazelcast	Hazelcast IMDG	In-memory data grid	hazelcast.com/products
iMatix Corporation	ZeroMQ	Message queue	zeromq.org
InfluxData	InfluxData	Time series database	influxdata.com
Ionic	Drifty	Cross-platform mobile framework	ionicframework.com
iText Group	iText	PDF manipulation in Java	itextpdf.com
Jfrog	Artifactory	Binary repository manager	jfrog.com/artifactory
jQuery Foundation	jQuery	JavaScript library	jquery.com
jQuery Foundation	jQuery CDN	JavaScript CDN	code.jquery.com
Junit	Junit	Unit testing framework	junit.org
Kubernetes	Kubernetes	Container management	kubernetes.io
Laravel	Akka	PHP web framework	laravel.com
Lightbend	Akka	Java implementation of actor models	akka.io

COMPANY	PROJECT	TYPE	WEBSITE
Lightbend	Lagom	Reactive microservices framework (Java, Scala)	lightbend.com/lagom
Lightbend	Play	Java and Scala web framework	playframework.com
MariaDB	MariaDB TX	DBMS	mariadb.com
Mesosphere	Marathon	Container orchestration	mesosphere.github.io/marathon
Meteor Development Group	Meteor	JavaScript framework	meteor.com
Microsoft	CNTK (Cognitive Toolkit)	Machine learning platform	microsoft.com/en-us/cognitive-toolkit
Microsoft	Xamarin	C# mobile app development	xamarin.com
MongoDB	MongoDB Community Server	NoSQL document database	mongodb.com/download-center#community
Mozilla	Bugzilla	Bug tracking system	bugzilla.mozilla.org
Nagios	Nagios Core	Infrastructure monitoring	nagios.org/projects/nagios-core
Nagios	Nagios Xl	APM, infrastructure monitoring, network monitoring, FEO, ITOA	nagios.com/products/nagios-xl
Netflix	Hystrix	Latency and fault tolerance library	github.com/Netflix/Hystrix
Netflix	Ribbon	Load balancing library	github.com/netflix/ribbon
NGINX	nginmesh	Service mesh	github.com/nginmesh/nginmesh
Node.js Foundation	Node.js	JavaScript environment	nodejs.org
npm, inc.	npm	Package manager	npmjs.com
NumFocus	NumPy	"Mathematical computing library (i.e. multidimensional arrays, linear algebra, Fourier transforms) for Python"	numpy.org
Open Source Matters	Joomla!	CMS	joomla.org
OpenESB	OpenESB	ESB	open-esb.net
OpenStack	OpenStack	Virtual machines	openstack.org
Oracle	MySQL Community Edition	RDBMS	mysql.com

COMPANY	PROJECT	TYPE	WEBSITE
Oracle	Glassfish	Java application server	javaee.github.io/glassfish/download
OWASP	OWASP Projects	Various open source security projects	owasp.org/index.php/Category:OWASP_Project
Pivotal	Greenplum	Open source data warehouse and analytics	pivotal.io/pivotal-greenplum
Pivotal	RabbitMQ	Message queue	network.pivotal.io/products/pivotal-rabbitmq
Pivotal	Spring Integration	Integration framework	projects.spring.io/spring-integration
Pivotal	Spring Framework	Enterprise Java library	projects.spring.io/spring-framework
Pivotal	Spring Cloud	Distributed systems framework	cloud.spring.io
Polymer Authors	Polymer	Web components library	polymer-project.org
PostgreSQL	PostgreSQL	RDBMS	postgresql.org
Progress Software	NativeScript	Cross-platform mobile framework	nativescript.org
Prometheus	Prometheus	Application monitoring	prometheus.io
Puppet Labs	Puppet	Infrastructure automation/configuration management	puppet.com
Raspberry Pi Foundation	Raspberry Pi	Open source prototyping platform	raspberrypi.org
Red Hat	JBoss Enterprise Application Platform	App containers, application platform-as-a-service	redhat.com/en/technologies/jboss-middleware/application-platform
Red Hat	Hibernate	ORM	hibernate.org
Red Hat	Ansible Tower	Configuration management and release automation	ansible.com
Red Hat	Fabric8	Integration platform-as-a-service	fabric8.io
Red Hat	HornetQ	Message queue	hortnetq.jboss.org
Red Hat	JBoss Fuse	ESB	redhat.com/en/technologies/jboss-middleware/fuse
Red Hat	JBoss EAP	Jakarta EE Platform	developers.redhat.com/products/eap/overview
Red Hat	WildFly	Java application server	wildfly.org

COMPANY	PROJECT	TYPE	WEBSITE
Red Hat	CoreOS Fleet	Container runtime system	github.com/coreos/fleet
Red Hat	CoreOS Flannel	Container orchestration	coreos.com/flannel/docs/latest
Red Hat	CoreOS Etcd	Container-defined networking	coreos.com/etcd
Redis Labs	Redis	In-memory database	redis.io
Redmine	Redmine	Project management application	redmine.org
Ruby on Rails	Ruby on Rails	Web framework	rubyonrails.org
SaltStack	Salt	Configuration management	saltstack.com
Selenium	Selenium WebDriver	Automated web testing	seleniumhq.org
Sonatype	Nexus	Repository management	sonatype.com/nexus/repository-sonatype
SQLite	SQLite	RDBMS	sqlite.org
StormMQ	StormMQ	Message queue	github.com/stormmq
Sysdig	Sysdig Falco	Behavioral activity monitor with container support	sysdig.com/falco
Telerik	KendoUI	Web framework	telerik.com
TensorFlow	TensorFlow	Machine learning framework	tensorflow.org
TravisCI	TravisCI	Continuous integration	travis-ci.org
Twitter	Bower	Package manager	bower.io
Vaadin	Vaadin	Server-side Java to HTML5	vaadin.com
Vue.js	Vue.js	JavaScript framework	vuejs.org
WebRTC	WebRTC	Real-time communication through APIs	webrtc.org
Yammer	Dropwizard	REST web services framework	dropwizard.io
Zipkin	Zipkin	Distributed tracing	zipkin.io

GLOSSARY

APACHE FOUNDATION
A nonprofit organization founded in 1999 that oversees development and hosting for hundreds of open-source projects, including Kafka, Hadoop, Apache Web Server, and more.

BRANCH
A duplicate of a piece of code that is currently in version control, so changes can be made without affecting the main source code.

COMMITTER
A member of the community that contributes to the code of an open-source project in the form of new features, bug fixes, or performance improvements.

COMMUNITY
A group of users that use or contribute to an open-source project.

COMMUNITY GUIDELINES
A set of rules decided on by community moderators of an open-source project, usually related to how contributors add code or disallowing negative treatment of other community members.

ECLIPSE FOUNDATION
A nonprofit organization founded in 2004 as a steward of the Eclipse IDE community, which has since expanded to include other open-source projects including Jakarta EE (formerly Java EE).

FORK
A project that was originally based on another project. Users may fork a project if they want to modify it for their own personal use, or drastically change it. For example, CI/CD tool Jenkins was developed as a fork of Hudson.

GIT
A version control system created to track changes made to source code by multiple users.

GITHUB
A web application for hosting code that is version-controlled using Git.

GNU GENERAL PUBLIC LICENSE
A widely used software license, first used by the GNU operating system, that makes it explicitly clear by the software's copyright holder that the software can be used and modified by anyone.

GOVERNANCE
The process in which a business or team manages the adoption of open-source software into existing projects or workflows.

LICENSE
A declaration of how a piece of software is to be used. In open-source software, licenses often make it legally explicit that the software can be used or modified by anyone.

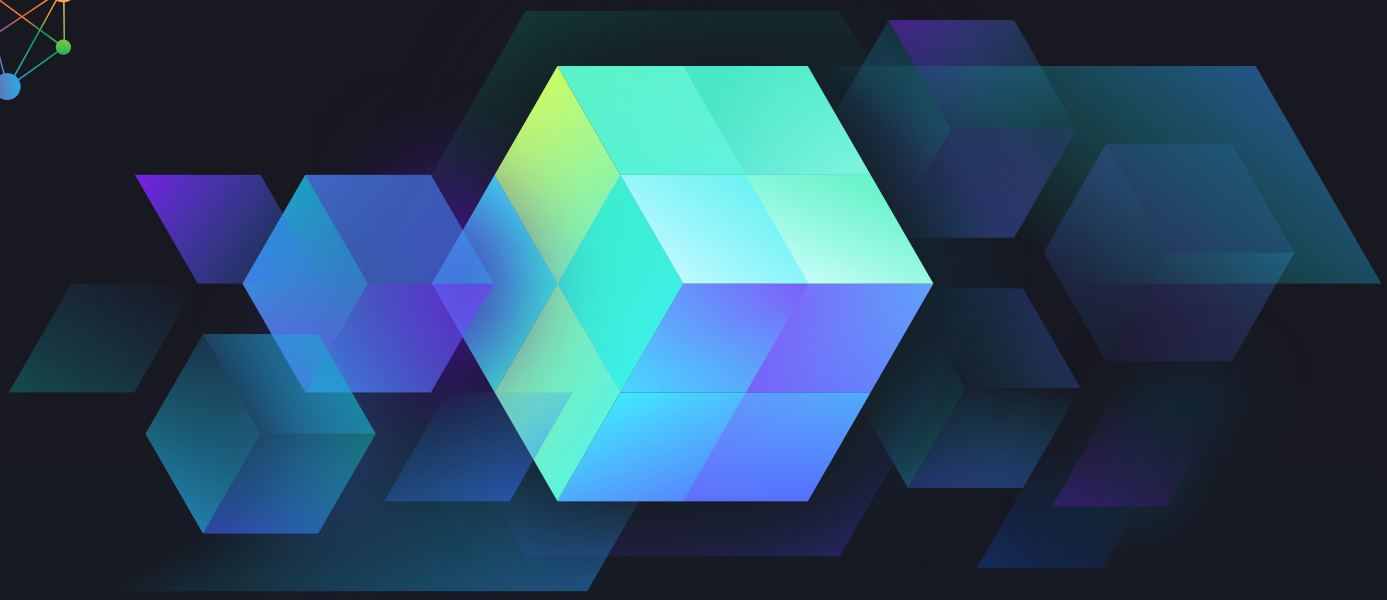
LINUX
An open-source operating system built on the Linux kernal created by Linus Torvalds and components of the open-source GNU operating system. It is the leading operating system installed on servers and mainframes, as well as the leading general-purpose OS due to the kernal's use in Android phones.

OPEN DATA
Data that is freely available to anyone to use, republish, analyze, and republish without copyright infringement.

OPEN-SOURCE SOFTWARE
Software that is freely available to use and modify by anyone. This is typically codified by using an open-source license.

PULL REQUEST
A notification from a committer that explains what changes have been made to source code, particularly as part of a GitHub repository.

SOURCE CODE
Code that is written by a programmer to be executed by a computer. All software is created by source code, but source code is not always included with software when it ships.



INTRODUCING THE

Microservices Zone

Start Implementing Your Microservice Architecture and Scale Your Applications

Whether you are breaking down existing monolithic legacy applications or starting from scratch, see how to overcome common challenges with microservices.

Keep a pulse on the industry with topics like: Java Microservices Tutorials and Code Examples, Best Practices for a REST-Based Microservices Architecture, and Patterns and Anti-Patterns to Get Started.

Visit the Zone

REST-BASED MICROSERVICE ARCHITECTURE

PATTERNS AND ANTI-PATTERNS

INTER-SERVICE COMMUNICATION

APPLICATIONS FOR CONTAINERS