Open Source

17-313 Fall 2025

Foundations of Software Engineering

https://cmu-17313q.github.io

Eduardo Feo Flushing

Outline

- What's open source?
- Historical context: How did we get here?
- Does it make sense to release open-source software?
- Good practices:
  - Being part of the community
  - Building an open source community

Motivation to understand open source.

- Companies work on open-source projects.
- Companies use open-source projects.
- Companies are based around open-source projects.
- Principles percolate throughout industry.
- Political/philosophical debate, and being informed is healthy.

What is Open Source Software?

Open-Source Software Defined

- The simplest definition is that it is software that is released under an “open source license”
- The Open Source Definition (OSD) describes 10 traits a software license must support [https://opensource.org/osd/]
- The OSD is maintained by the Open Source Initiative, a non-profit that supports the email discussion groups that judge licenses against the OSD
- There are around 80 OSI-approved licenses, but only a few are commonly used.

  [MIT License, Apache License aka ASLv2, GPLv2/3, Berkley or BSD, Mozilla or MPLv2, Eclipse License, Artistic License]

The OSD – Open Source Definition
(Attributes of the Software License)

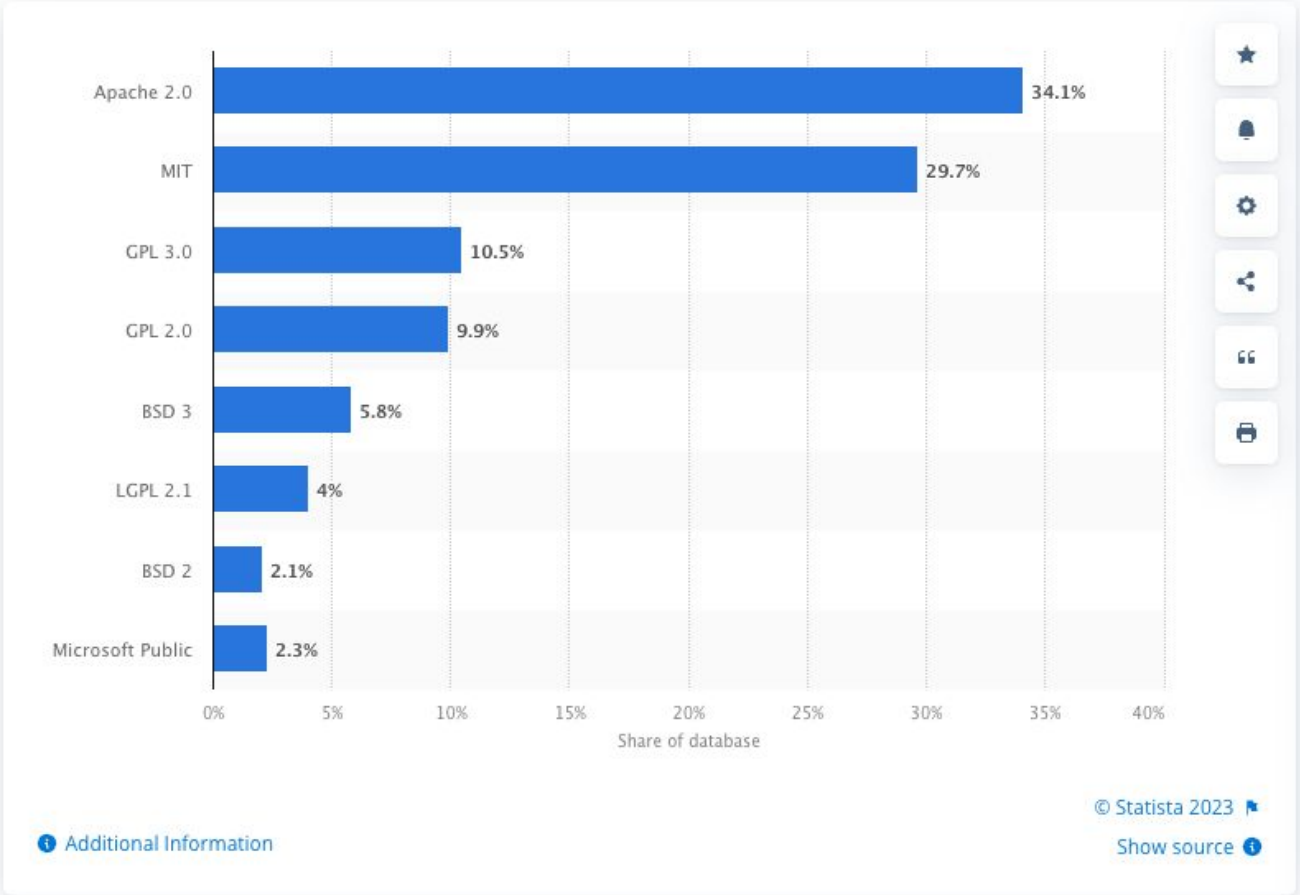
1. Free Redistribution
2. Source Code
3. Derived Works
4. Integrity of the Author’s Source Code
5. No Discrimination Against Persons or Groups
6. No Discrimination Against Fields of Endeavor
7. Distribution of the License
8. License Must Not Be Specific To A Product
9. License Must Not Restrict Other Software
10. License Must Be Technology Neutral


SOFTWARE LICENSE
AGREEMENT
THIS AGREEMENT is made and entered into this _____ day of
referred to as "Seller") and _____, 20__, by and between _____, (hereinafter
"Purchaser");
WITNESSETH:
WHEREAS, the Seller

Why learn about licenses?

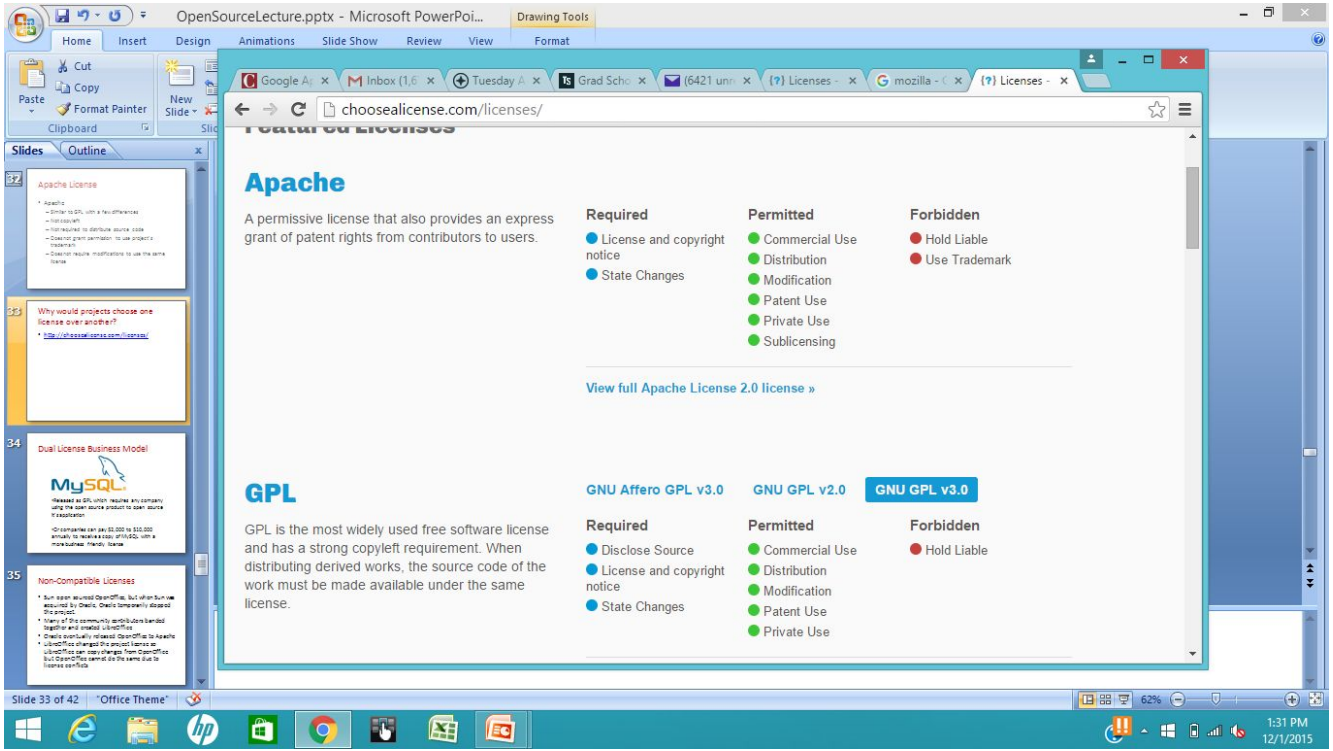
- Specific licenses may provide competitive advantages
- You may eventually want to release open-source software or become more involved in an open-source project
- Avoid legal issues

Most popular open source licenses worldwide in 2021



https://www.statista.com/statistics/1245643/worldwide-leading-open-source-licenses/

Which license to choose?



GNU General Public License: The **Copyleft** License

- Nobody should be restricted by the software they use. There are four freedoms that every user should have:
  - the freedom to use the software for any purpose,
  - the freedom to change the software to suit your needs,
  - the freedom to share the software with your friends and neighbors, and
  - the freedom to share the changes you make.
- Code must be made available
- Any modifications must be re-licensed **under the same license (copyleft)**

GPL 2.0 and 3.0 – Addresses free software problems

- **GPLv2**
  - Court ruling cannot nullify the license and if a court decision and this license contradict in distribution requirements, then the software cannot be distributed
  - Implicitly grants users a license to any patents held by contributors that are required to use the software.
- **GPLv3**
  - Clarifications on patent grants
  - Prevents “Tivoization” (Hardware restrictions)
- WARNING: Compatibility issues between both

LGPL

- Software must be a library (component to be used in other products)
- Similar to GPL but no copyleft requirement

BSD License

- No liability and provided as is.
- Copyright statement must be included in source and binary
- The copyright holder does not endorse any extensions without explicit written consent

MIT License

- Must retain copyright credit
- Software is provided as is
- Authors are not liable for software
- Unlike the GPL, it does not require that derivative works be open source
- No other restrictions
  - (including closing the source) :(

Apache License

- Similar to MIT with a few differences
- Not copyleft
- Not required to distribute source code
- Does not grant permission to use project's trademark
- Does not require modifications to use the same license
- Allows proprietary use
- Explicit grant of patent rights

Two extremes

- No License (default is “all rights reserved”)
- Public Domain Dedication (e.g., Unlicense)

Activity: Choose the Appropriate License (small groups)

- Analyze a given software scenario and select the most appropriate license from the following: GPL, LGPL, MIT, BSD, Apache.
- Read the provided scenarios and its goals (e.g., commercial use, community contributions, patent protections).
- Consider the key factors: Discuss with your group the following aspects:
  - Does the project need copyleft protections or permissive terms?
  - Are there concerns about proprietary use or redistribution?
  - Is patent protection important?
- Choose the license that best aligns with the scenario requirements and justify your decision.

You can consult **choosealicense.com**

You are creating a software tool and want others to freely use, modify, and even sell it without many restrictions.

What type of open-source license would you choose?

You have a commercial (proprietary) app and but want to use an open-source component. How can you make sure you don’t accidentally have to open-source your entire app?

You want to release your software as open source but you also want to protect yourself from future patent lawsuits.

What kind of license or clause should you include?

You want your open-source project to spread widely into both open-source and commercial products but you still want to get credit for your work.

How would you license it?

You are building a piece of open-source software, and you want to make sure that if someone improves it and shares it,

they also have to share their improvements openly with everyone.

What kind of license would you choose?

You want people to freely use and modify your software, but you don’t want them to use your name or your company’s name to promote their versions without your permission.

How would you set up your license?

You are combining open-source components with your own proprietary modules.

How can you make sure you stay legally safe while keeping parts of your code private?

Dual License Business Model

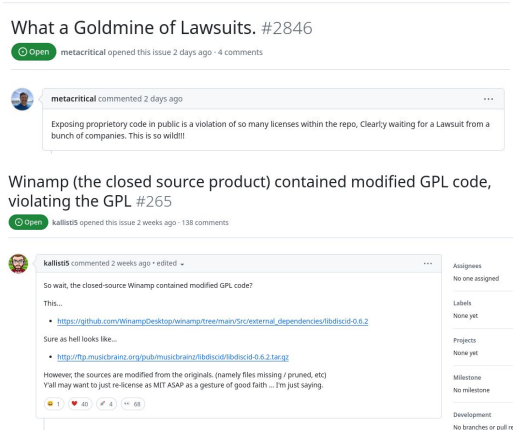
- Released as GPL which requires a company using the open source product to open source it's application
- Or companies can pay $2,000 to $10,000 annually to receive a copy of MySQL with a more business friendly license

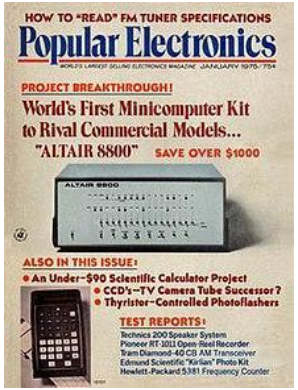
Risk: Incompatible Licenses

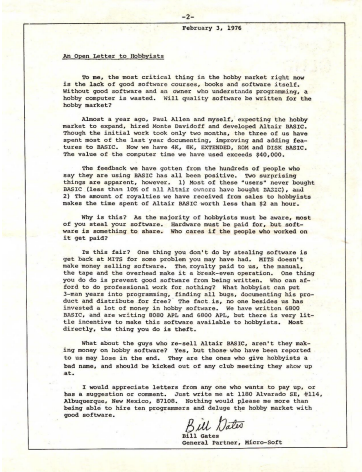
- Sun open sourced OpenOffice, but when Sun was acquired by Oracle, Oracle temporarily stopped the project.
- Many of the community contributors banded together and created LibreOffice
- Oracle eventually released OpenOffice to Apache
- LibreOffice changed the project license so LibreOffice can copy changes from OpenOffice but OpenOffice cannot do the same due to license conflicts

What not to do

- **Winamp** was a popular media player in the late 1990s and early 2000s
  - Open-source its code to revive community interest and foster further development and innovation.
- Source code release inadvertently included proprietary components from Microsoft, Intel, Dolby, and the SHOUTcast server software
- The initial release under the *"Winamp Collaborative License"* imposed restrictions that **contradicted** open-source principles
  - Prohibiting forks and distribution of modified versions
- Discovery of GPL Code:
  - After releasing the Winamp source code, it was found to include GPL-licensed components.







-2-

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds $40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than $2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

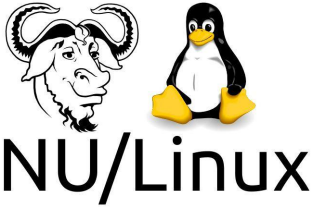
Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

Bill Gates
General Partner, Micro-Soft

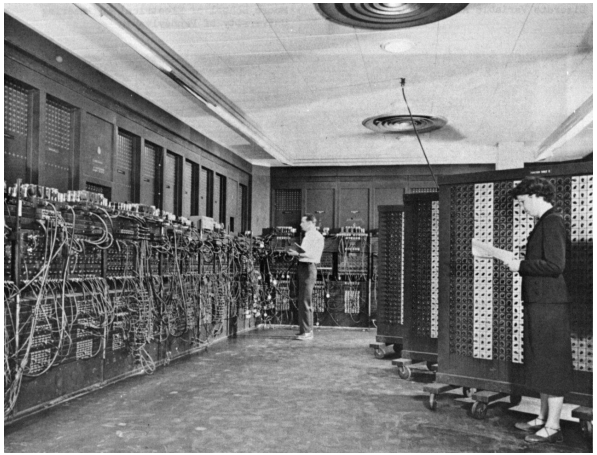


How did we get here?

A bit of history to understand why open source looks the way it does today

A bit of history: 1950s

- Grace Hopper writes the world's first compiler A-0
- IBM introduces FORTRAN
- Software released as public domain



A bit of history: 1960s

- IBM's System/360 (general-purpose computing)
- The rise of "minicomputers"
  - DEC PDP-8
- "IBM and the seven dwarfs"
- Software usually bundled with hardware
- Software development is very expensive
- 1969: **"Software unbundling"**
  - **Software Industry is born**

A bit of history: 1970s

- Rise of the “microcomputers”
  - PDP, Altair
- Unix first release by Bell Labs
- Microsoft was founded
- Gate’s “Open Letter to Hobbyists”

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds $40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than $2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

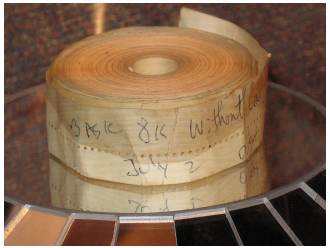
Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

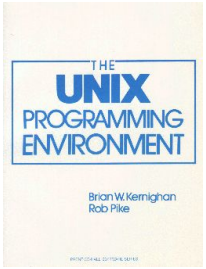
Bill Gates
General Partner, Micro-Soft

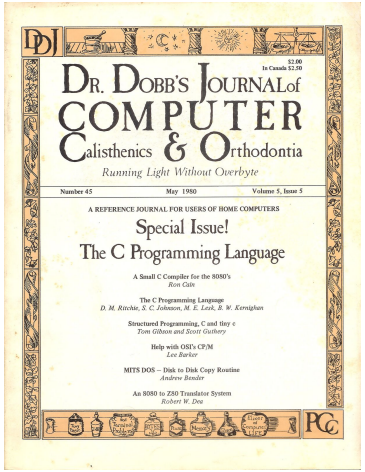


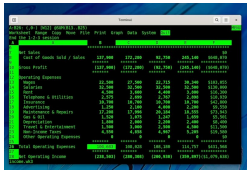
```
;*****************************************
;
;     TINY BASIC FOR INTEL 8080
;           VERSION 2.0
;         BY LI-CHEN WANG
;       MODIFIED AND TRANSLATED
;          TO INTEL MNEMONICS
;          BY ROGER RAUSKOLB
;           10 OCTOBER 1976
;             @COPYLEFT
;        ALL WRONGS RESERVED
;
;*****************************************
```

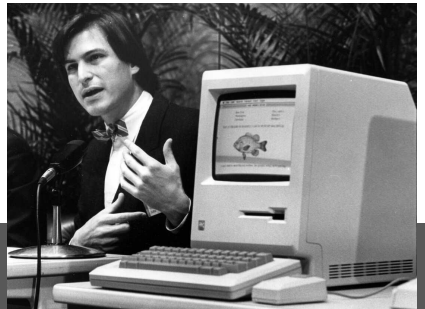
A bit of history: 1980s

- Rise “personal computers”
- Software industry flourishes
  - Adobe, Autodesk, Lotus, …
- Informal software sharing
  - Software “hobbyists”
- Apple Computer, Inc. v. Franklin Computer Corp.
- The *Unix Programming Environment* is published
- Stallman starts the **Free Software Movement**









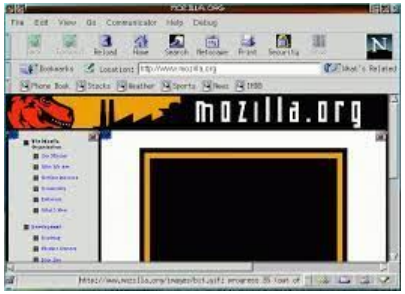
“Free as in free speech.”

GNU/Linux

FREE SOFTWARE FOUNDATION

A bit of history: 1990s

- Linux
- Open Source Initiative
- The First Browser War
- Netscape started the open source Mozilla project.



Open Source Initiative

Anchor

Do you want to support Open Source? Book a meeting with us now.

Innovator

DataStax

Premier

Bloomberg Engineering, cisco, indeed, Red Hat, Google

Maintainer

CapitalOne, COMCAST, FerretDB, GitHub, Microsoft, OpenLogic by Perforce, salesforce, slim.ai, TIDELIFT

Supporter

eye/o, harness, ENGINEERING

open source initiative®

Old Perception:

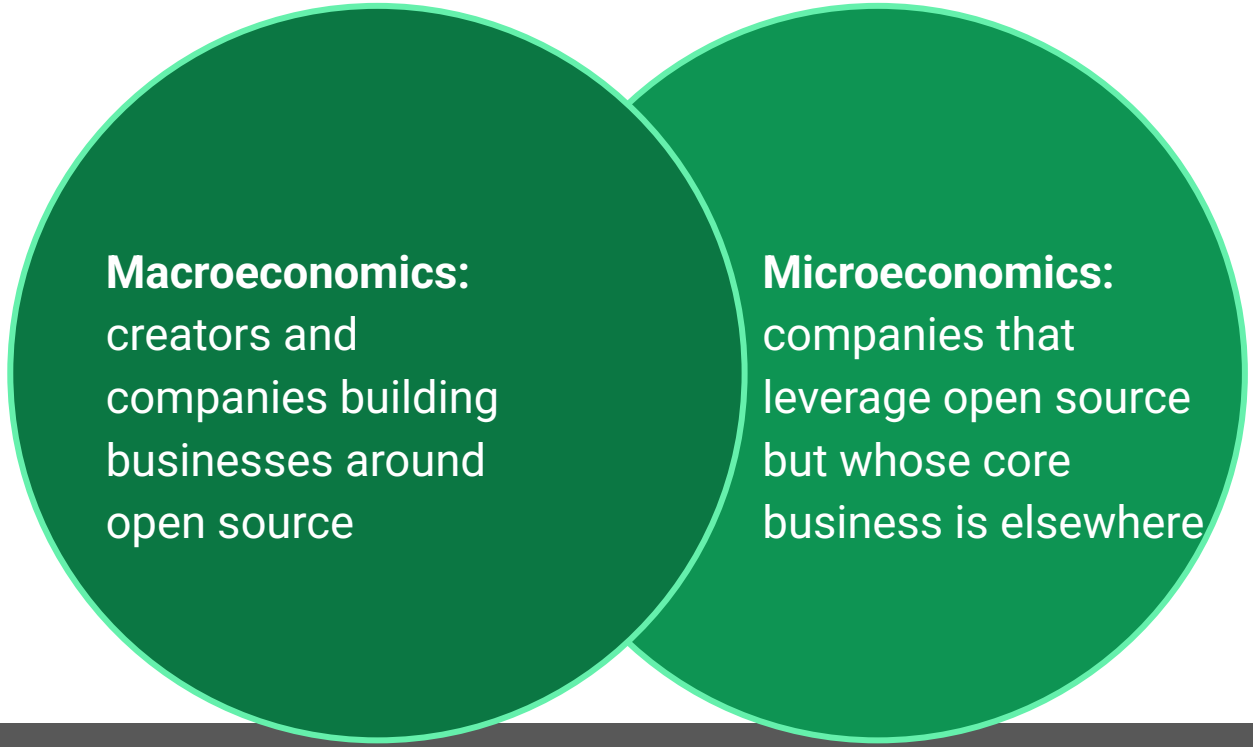
- Anarchy
- Demagoguery
- Ideology
- Altruism
- Many eyes

Does it make sense to release open source software?

why would companies invest millions into it?

Framing Open Source Economics

Macroeconomics vs. Microeconomics



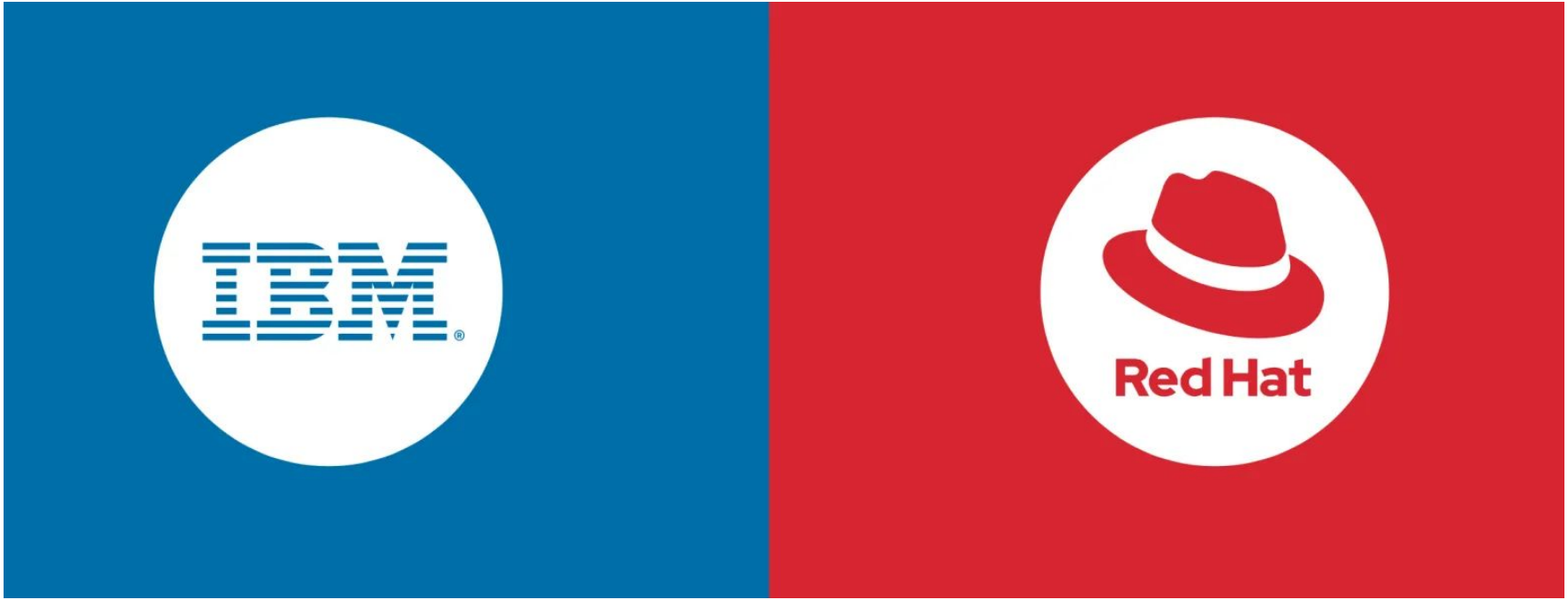
The **Macroeconomics**: Open Source as a Business

- Professional services
  - RedHat, Canonical, ..
  - Consulting, training
  - Problems: Ease of use dilemma, renewal rates, service-only competition

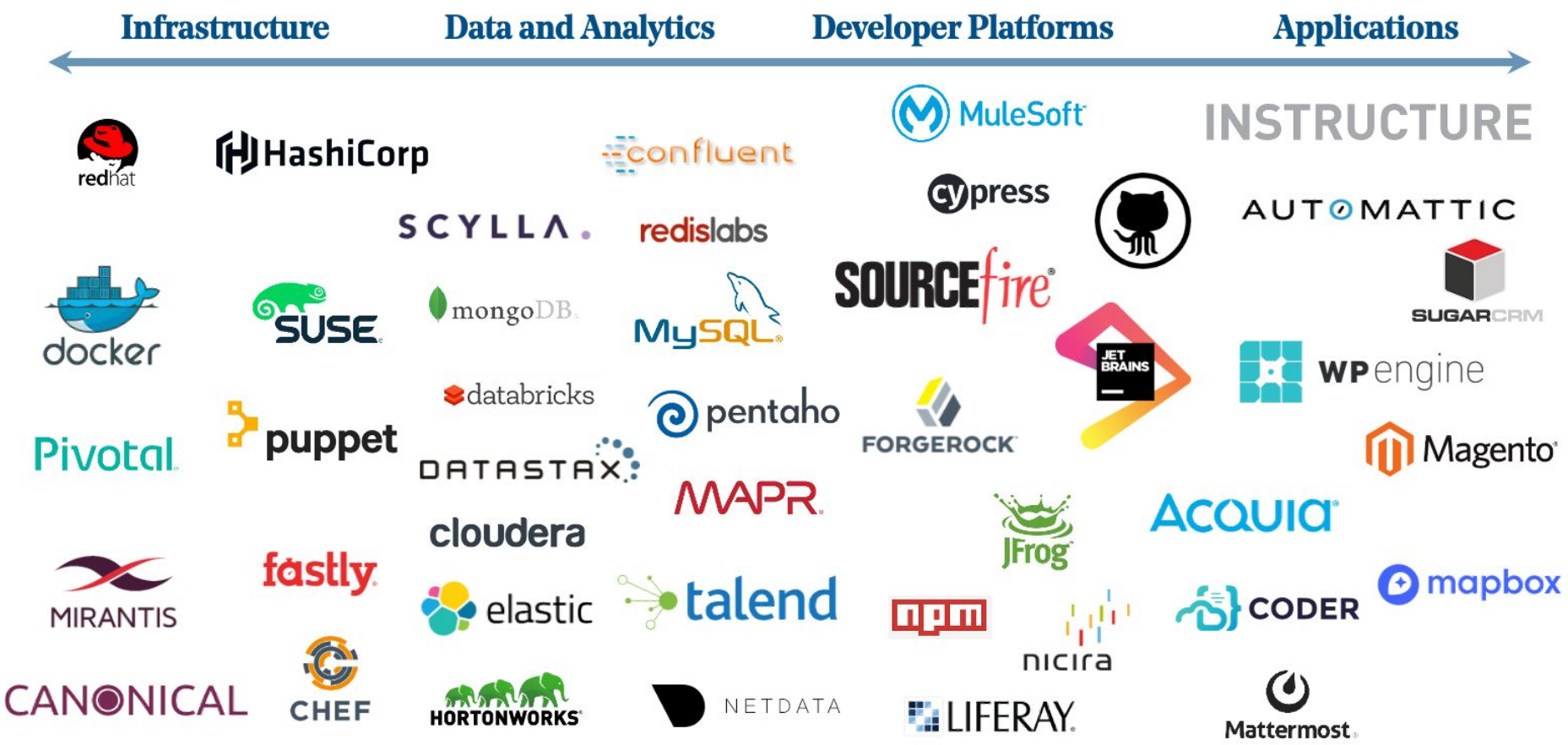
- Open core
  - GitLab, Databricks (Apache Spark), MongoDB
  - Problem: Competition on the commercial offerings. E.g., tools

- Software as a service
  - WordPress
  - Problem: Free riding by cloud providers

Red Hat: A successful example

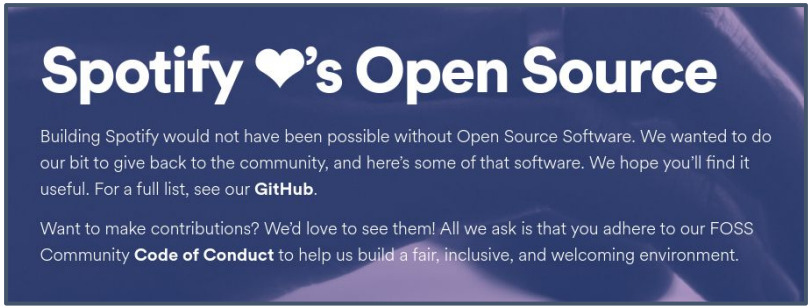


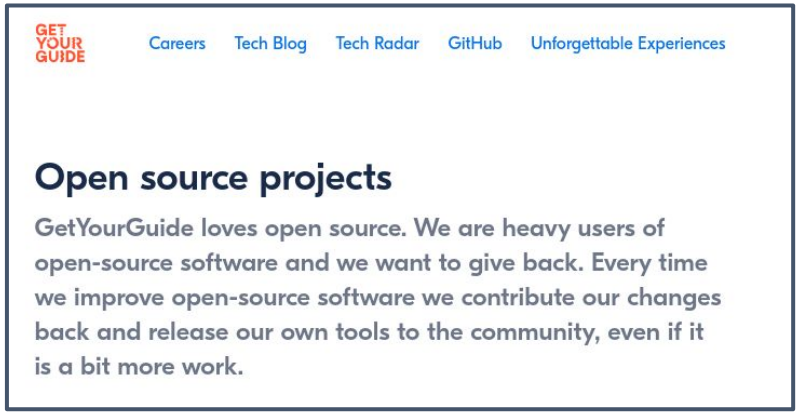
https://www.fundinguniversity.com/company-histories/red-hat-inc-history/

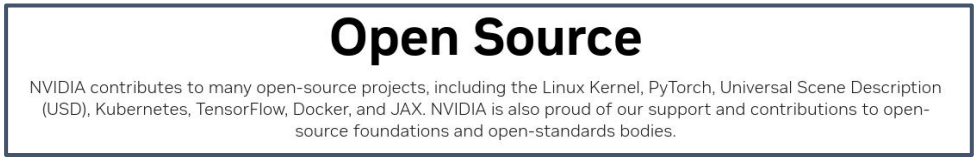


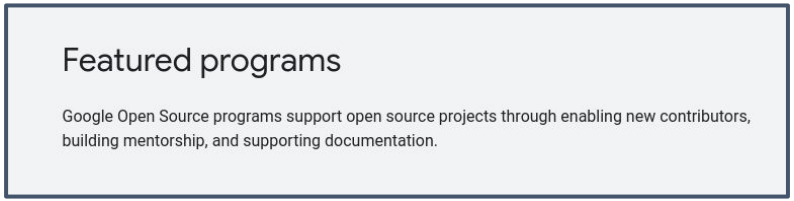
source : https://www.bvp.com/atlas/roadmap-open-source

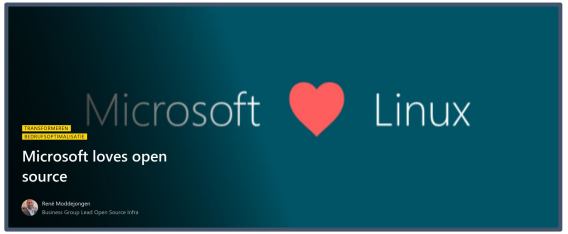
Is it Altruism?

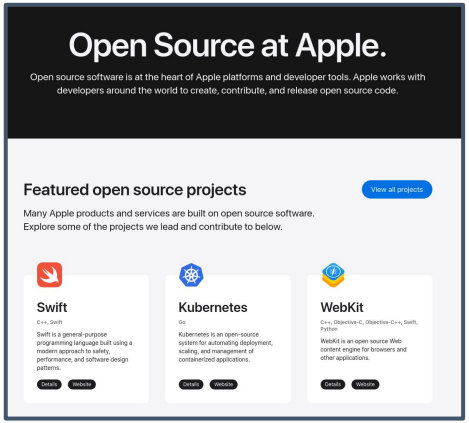












Open Source Production Economics (Why Publish? Why Build A Community)

- Kubernetes = 4.6M Lines of Code = ~$630M
- VS Code = 1.2M Lines of Code = ~$90M
- Each of the communities have an architecture of participation where 50% of the contributions come from OUTSIDE the primary owner (respectively Google and Microsoft)

Rule of Thumb: 1MLoC = 100 staff * 5 years = $75M ($100K base * 1.5 gross up)

The **Microeconomics**: Open Source as a Strategic Business Enabler

Even if open source is not your business model, strategic use and contribution can become major competitive advantages

The **Microeconomics**: Build, Buy… or Something Smarter?

- Build versus buy is the most fundamental question in engineering
- The cost of build-from-scratch vs the cost of buying/licensing
- Add in the cost of maintenance over time

Open source adds a third option to build vs buy: borrow and share

Borrow-and-Share

Add OSI-licensed projects and you add **borrow-and-share** to decision

- **Borrowing** the best that communities have built.
- **Adapting** it to your needs.
- **Sharing** enough to sustain it and keep moving faster.

Leverage freely available, community-maintained technology, adapt it to your needs, contribute improvements back, and benefit from the evolution of the broader ecosystem.

Total Cost of Ownership

Cost Category	Proprietary	Open Source
Licensing	High	None/Low
Implementation	Medium	High
Maintenance	Medium	Medium
Upgrades/Patches	Vendor-driven	Self-managed
Staff Training	Vendor-provided	Internal teams
Security & Compliance	Vendor coverage	Internal responsibility

~~Total Cost of Ownership~~
Total Power of Collaboration

Not Just DIY

Leverage the community for fixes and innovation

Developer Growth

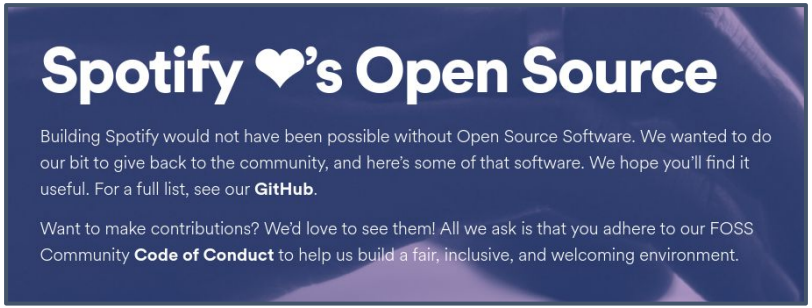
Developers grow faster through real-world collaboration

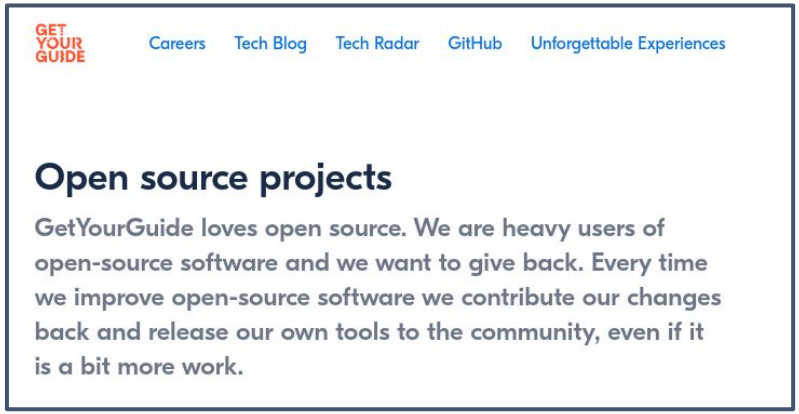
Hidden Costs of Closed Code

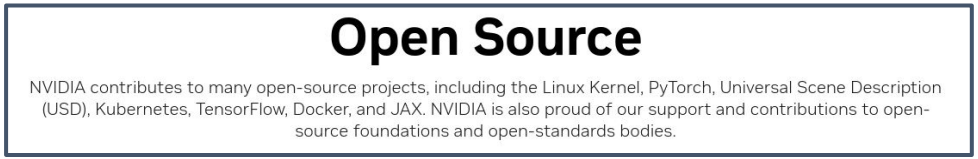
Contracts, NDAs, and partner negotiations become a burden without openness

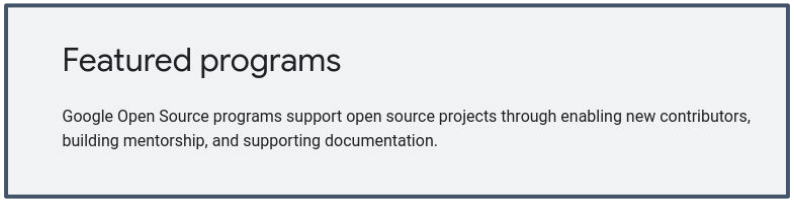
Open Source transforms ownership into collaboration, and cost into investment

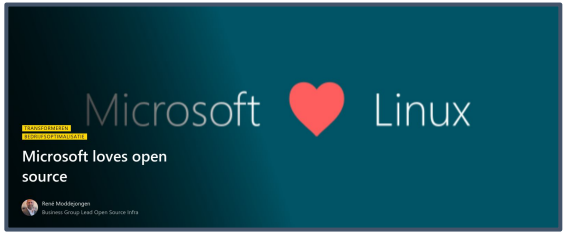
Is it Altruism?

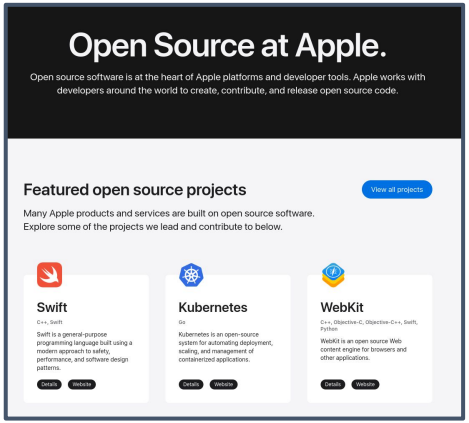












Is it Altruism?

- It isn't about altruism – it's about economics
- It is economically an easier decision to use (borrow)-and-share than build or buy
- You always get more than you give economically

From Consumers to Contributors: How Companies are Driving Open Source Innovation

- Airbnb: Published Apache Superset, now a leading BI tool for data visualization.
- Lyft: Open-sourced Envoy, a high-performance service proxy, powering service meshes like Istio.
- Google: Published Kubernetes, now the de facto standard for container orchestration.
- Netflix: Contributed Spinnaker, a multi-cloud continuous delivery platform.
- Meta (Facebook): Released PyTorch, one of the most popular AI/ML frameworks.
- Twitter: Published Apache Heron, a real-time stream processing engine.
- Databricks: Contributed to Apache Spark, enabling scalable data analytics.
- Shopify: Contributed improvements to Ruby on Rails, ensuring scalability for e-commerce applications.

Personal Economics

- Having one's name on key contribution streams in an open source community world is **some of the best resumé content** one can have:
  - to get work done,
  - to work in a collaborative engineering setting,
  - to demonstrate your understanding of a technology base.
- The connections you make in a community expands your network, and it's the weak connections that make a difference in job market
- Reading software is a good way to improve one's skills, so reading known good software projects is an opportunity to improve skills.

Granovetter, Mark "The Strength of Weak Ties", Stable URL:
https://www.jstor.org/stable/2776392

Activity: Build vs Buy vs Borrow & Share

In groups, discuss the following engineering scenarios.

For each scenario, write down your choice: Build, Buy, or Borrow & share (Open Source)

Optional: Write one sentence explaining why you chose it.

There is no single correct answer, the goal is to justify your choice.

Scenario 1: User Authentication

You need login, signup, password reset, and OAuth integration for your app.

Build, Buy, or Borrow & Share (Open Source)?

Scenario 2: Custom ML Algorithm

You need a domain-specific machine learning model that doesn't exist anywhere else.

Build, Buy, or Borrow & Share (Open Source)?

Scenario 3: Payment Processing

Your app must process international credit card payments and handle compliance.

Build, Buy, or Borrow & Share (Open Source)?

Scenario 4: Internal Data Visualization Dashboard

Your team needs rich dashboards for internal analytics and monitoring.

Build, Buy, or Borrow & Share (Open Source)?

Open Source done right

What can you do now to get started?

Open Source Governance Styles

- Consensus
  - Apache
  - Rust
  - (many others)
- Dictator
  - Python
  - Linux
- Corporate

Contributing to Open Source: Skills

- Written communication
  - Email, chat, and design documents are core to asynchronous work
- Resilience and the ability to handle feedback openly
- Technical ability
- Political ability

Contribute early

- When you need to customize an open-source tool, try to contribute your changes back to the original project (the "upstream").
- Reduces the maintenance burden (you don't have to keep patching every time the project updates).
- Builds reputation and influence inside important open-source communities.

Version Control Discipline

What Sort of Contributions Can People Make?

Contributions

- Code–fixes and functionality increases code value and broadens use
- Bug reports indicate a new test path or use
- New configurations broadens the user base increasing use value
- Documentation (answering forum/email questions, creating tutorials, presentations, FAQ, etc.) broadens the user base
- Forum time answering questions
- Translations (messages, user document, etc.) can make a big difference
- ANYTHING SOMEONE BRINGS TO THE PROJECT

The Open Source Community: A Living Ecosystem

Sponsors

Fund, promote, sustain

Creators

Start projects, set vision

Users

Adopt, use, provide feedback

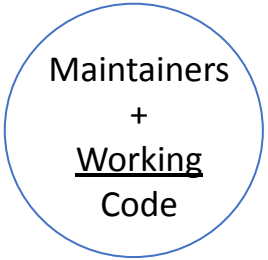
Maintainers

Review and approve changes

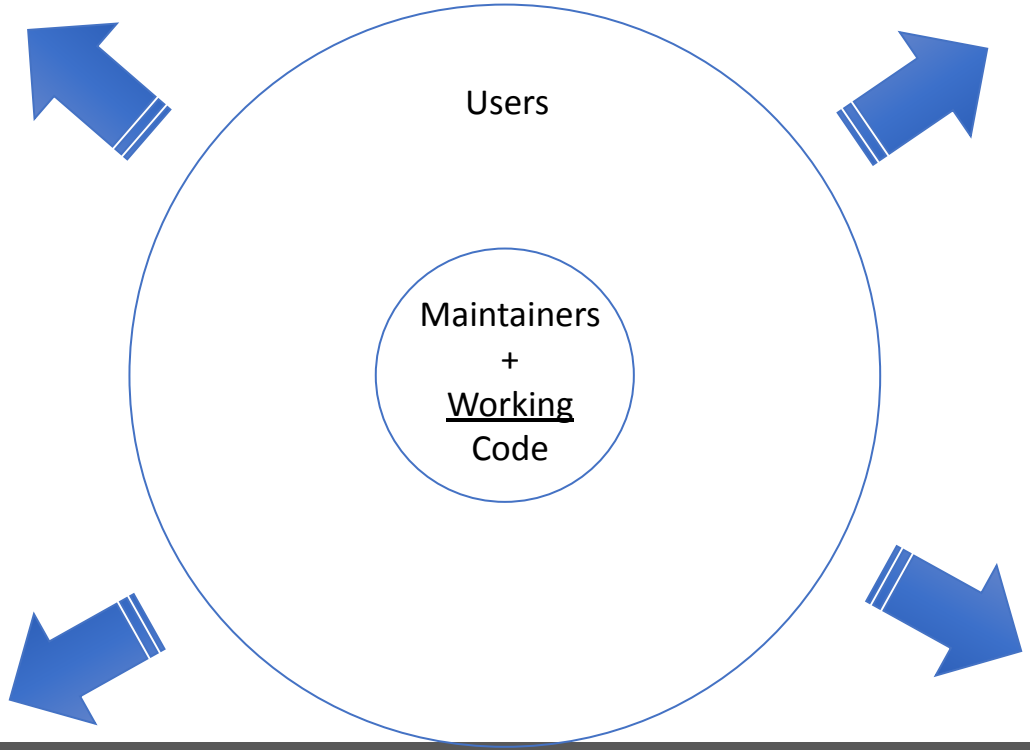
Contributors

Add features, fix bugs, improve documentation

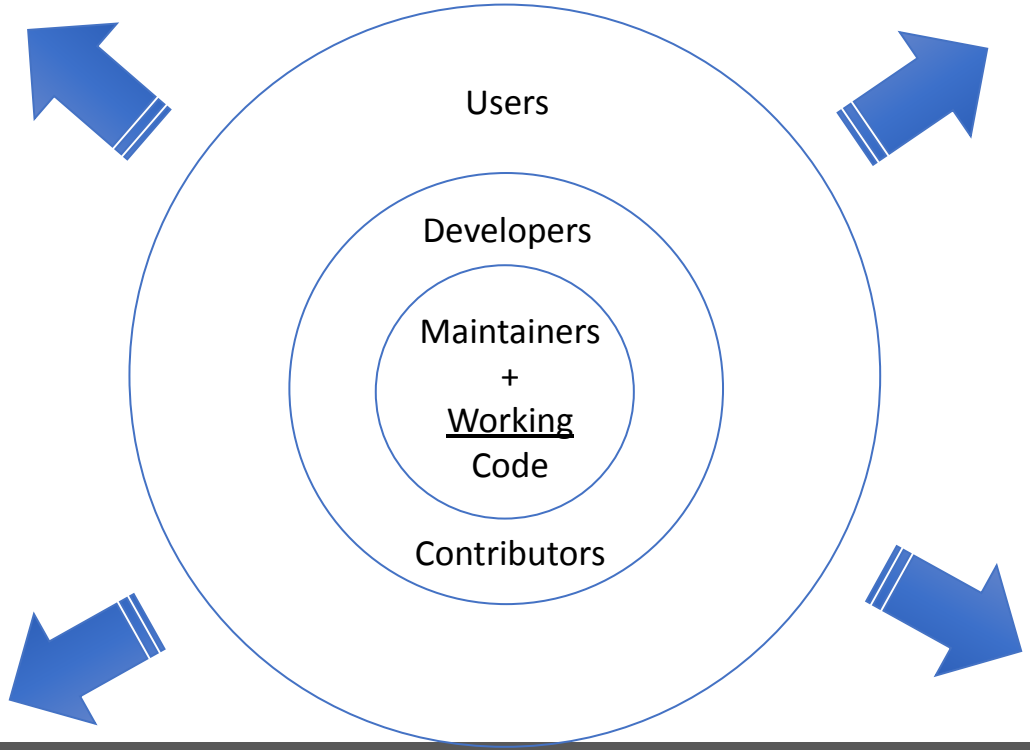
Three On Ramps Need to be Built



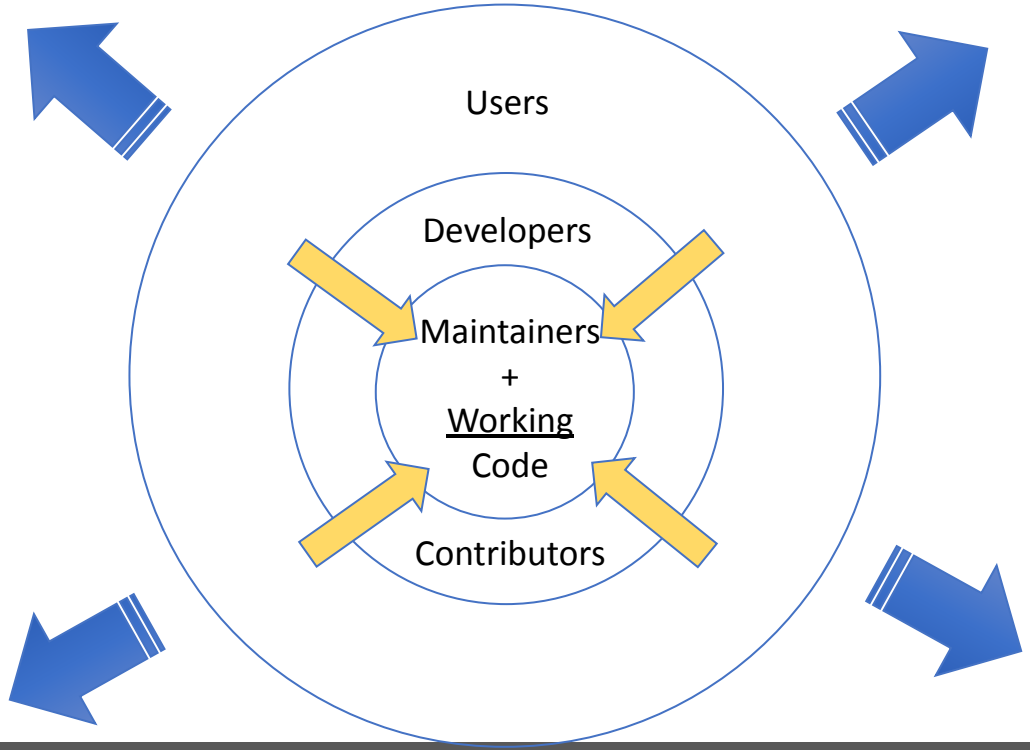
Three On Ramps Need to be Built



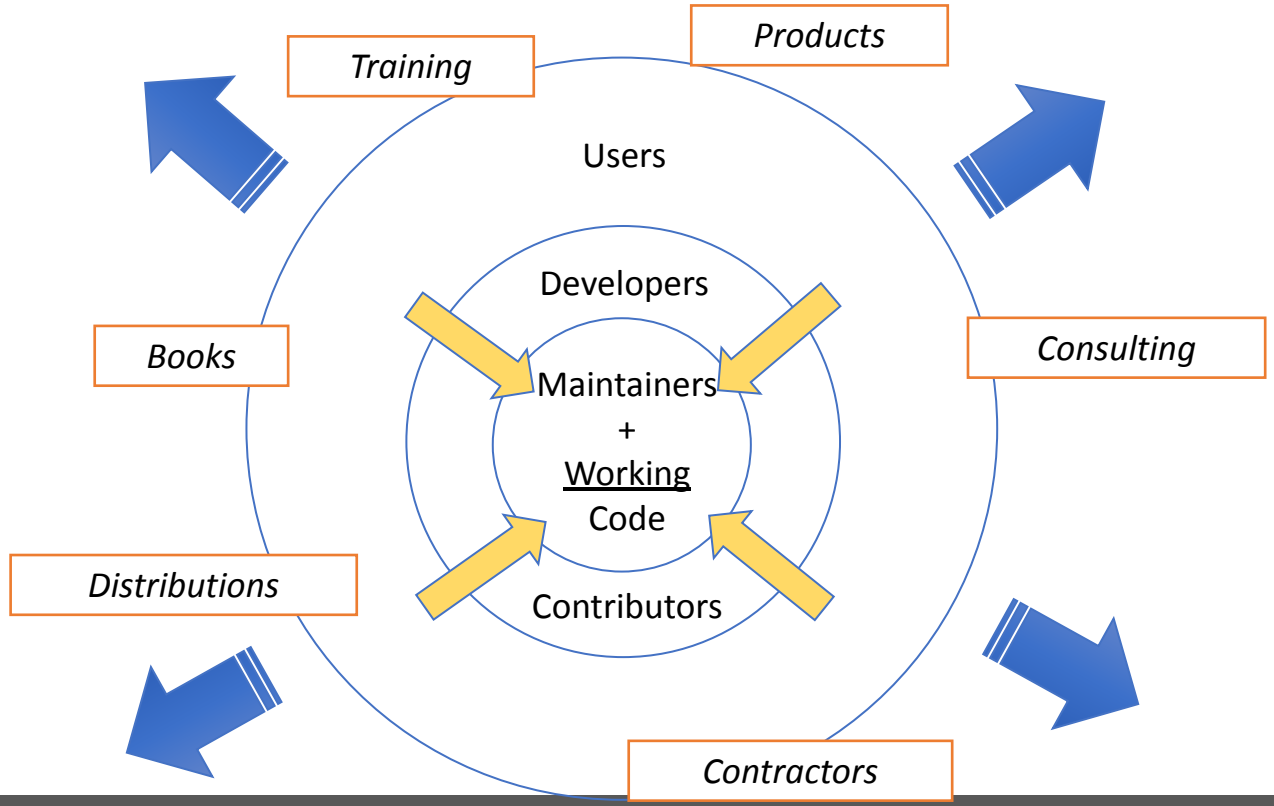
Three On Ramps Need to be Built



Three On Ramps Need to be Built



Three On Ramps Need to be Built



Three On Ramps for Community Building

How do you encourage people to use our project?

(Because that is where we will find bugs reports & developers)

How do you encourage developers selfishly to experiment?

(Because these are our future contributors)

How do you encourage developers to share their work?

(Because contribution flow is the growth and success of our project)

Three On Ramps for Community Building

How do you encourage people to use our project?

(How do we make it easy to install/configure/use the software?)

How do you encourage developers selfishly to experiment?

(How do we make it easy to build/test/experiment?)

How do you encourage developers to share their work?

(How do we make it easy to contribute?)

99% of Fortune 500 companies currently use open source software

80% of IT departments will increase their use of open source in 2021

35% of all enterprise software is based on open source code

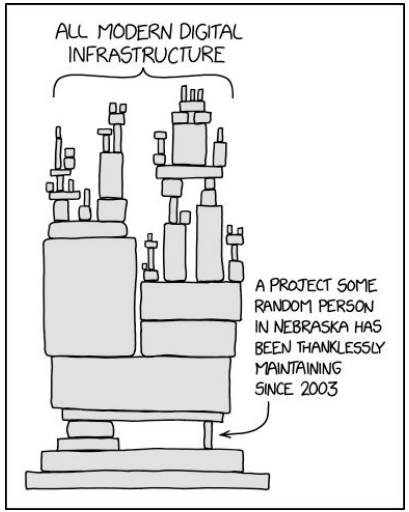
Over 56 million developers are contributing to open source projects

Over 140 million open projects are listed on GitHub[1]

10,000 lines of code are added to Linux every day, making it the fastest evolving software

Sources: GitHub, BCG analysis.

[1]GitHub is the leading platform for open source software collaboration.



Software Dependencies

Left-pad (March 22, 2016)

OBSESSIONS

QUARTZ

NPM ERR!

How one programmer broke the internet by deleting a tiny piece of code

THE VERGE TECH REVIEWS

REPORT TECH

How an irate developer briefly broke JavaScript

Unpublishing 11 lines of code brought down an open source house of cards

By Paul Miller | @futurepaul | Mar 24, 2016, 4:29pm EDT

SIGN IN

The Register

{* SOFTWARE *}

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

Left-pad (March 22, 2016)

npmjs.org tells me that left-pad is not available (404 page) #4

Closed **silkentrance** opened this issue on Mar 22, 2016 · 193 comments

silkentrance commented on Mar 22, 2016

When building projects on travis, or when searching for left-pad on npmjs.com, both will report that the package cannot be found.

Here is an excerpt from the travis build log

```
npm ERR! Linux 3.13.0-40-generic
npm ERR! argv "/home/travis/.nvm/versions/node/v4.2.2/bin/node" "/home/travis/.nvm/versions/node/v4.2.2/bin/npm
npm ERR! node v4.2.2
npm ERR! npm  v2.14.7
npm ERR! code E404
npm ERR! 404 Registry returned 404 for GET on https://registry.npmjs.org/left-pad
npm ERR! 404
npm ERR! 404 'left-pad' is not in the npm registry.
npm ERR! 404 You should bug the author to publish it (or use the name yourself!)
npm ERR! 404 It was specified as a dependency of 'line-numbers'
npm ERR! 404
npm ERR! 404 Note that you can also install from a
npm ERR! 404 tarball, folder, http url, or git url.
npm ERR! Please include the following file with any support request:
npm ERR!     /home/travis/build/coldrye-es/pingo/npm-debug.log
make: *** [deps] Error 1
```

And here is the standard npmjs.com error page https://www.npmjs.com/package/left-pad

However, if I remove left-pad from my local npm cache and then reinstall it using npm it will happily install left-pad@0.0.4.

88 3

Left-pad (Docs)

left-pad

String left pad

build unknown

Install

```
$ npm install left-pad
```

Usage

```
const leftPad = require('left-pad')

leftPad('foo', 5)
// => "  foo"

leftPad('foobar', 6)
// => "foobar"

leftPad(1, 2, '0')
// => "01"

leftPad(17, 5, 0)
// => "00017"
```

Install

```
> npm i left-pad
```

Repository
github.com/stevemao/left-pad

Homepage
github.com/stevemao/left-pad#readme

Weekly Downloads
2,962,641

Version	License
1.3.0	WTFPL

Unpacked Size	Total Files
9.75 kB	10

Issues	Pull Requests
3	7

Last publish
4 years ago

Left-pad (Source Code)

17 lines (11 sloc) | 222 Bytes

```
1  module.exports = leftpad;
2
3  function leftpad (str, len, ch) {
4    str = String(str);
5
6    var i = -1;
7
8    if (!ch && ch !== 0) ch = ' ';
9
10   len = len - str.length;
11
12   while (++i < len) {
13     str = ch + str;
14   }
15
16   return str;
17 }
```

See also: isArray

5 lines (4 sloc) | 133 Bytes

```
1  var toString = {}.toString;
2
3  module.exports = Array.isArray || function (arr) {
4    return toString.call(arr) === '[object Array]';
5  };
```

isarray

`Array#isArray` for older browsers and deprecated Node.js versions.

build passing downloads 227M/month

Just use Array.isArray directly, unless you need to support those older versions.

Usage

```
var isArray = require('isarray');

console.log(isArray([])); // => true
console.log(isArray({})); // => false
```

Install

```
> npm i isarray
```

Repository
github.com/juliangruber/isarray

Homepage
github.com/juliangruber/isarray

Weekly Downloads
50,913,317

Version	License
2.0.5	MIT

Unpacked Size	Total Files
3.43 kB	4

Issues	Pull Requests
4	3

Simple Node.js Code

https://github.com/MrDataScientist/Nodejs-10-projects-examples

npm install -g express

npm install -g express-generator

$ express express-website

$ cd express-website/

$ npm install

```
up to date, audited 122 packages in 629ms
19 vulnerabilities (3 low, 3 moderate, 8 high, 5 critical)
To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
```

How do software projects manage third-party dependencies on reusable libraries?

- It’s hard
- It’s mostly a mess (everywhere)
- But it’s critical to modern software development

What is a Dependency?

- A **dependency** is anything your code needs in order to **build, run, or be tested**.
- It can be another module, library, or service your code relies on.
- Build systems mainly track the code dependencies, but understanding runtime dependencies is equally important
- If a dependency isn't available or compatible, your system can't function correctly
  - *Just like missing ingredients or broken tools in a recipe*

Examples of dependency views

master NodeBB / install / package.json

Code Blame 201 lines (201 loc) · 6.13 KB

```
44          "autoprefixer": "10.4.20",
45          "bcryptjs": "2.4.3",
46          "benchpressjs": "2.5.1",
47          "body-parser": "1.20.3",
48          "bootbox": "6.0.0",
49          "bootstrap": "5.3.3",
50          "bootswatch": "5.3.3",
51          "chalk": "4.1.2",
52          "chart.js": "4.4.4",
53          "cli-graph": "3.2.2",
54          "clipboard": "2.0.11",
55          "colors": "1.4.0",
56          "commander": "12.1.0",
57          "compare-versions": "6.1.1",
58          "compression": "1.7.4",
59          "connect-flash": "0.1.1",
60          "connect-mongo": "5.1.0",
61          "connect-multiparty": "2.2.0",
62          "connect-pg-simple": "10.0.0",
63          "connect-redis": "7.1.1",
64          "cookie-parser": "1.4.6",
65          "cron": "3.1.7",
```

github.com/CMU-313/Teedy/blob/main/pom.xml

```
152     <dependencyManagement>
153       <dependencies>
154         <dependency>
155           <groupId>com.sismics.docs</groupId>
156           <artifactId>docs-core</artifactId>
157           <version>${project.version}</version>
158         </dependency>
159
160         <dependency>
161           <groupId>com.sismics.docs</groupId>
162           <artifactId>docs-web-common</artifactId>
163           <version>${project.version}</version>
164         </dependency>
165
166         <dependency>
167           <groupId>com.sismics.docs</groupId>
168           <artifactId>docs-web-common</artifactId>
169           <type>test-jar</type>
170           <version>${project.version}</version>
171         </dependency>
172
173         <dependency>
174           <groupId>com.sismics.docs</groupId>
175           <artifactId>docs-web</artifactId>
176           <version>${project.version}</version>
177         </dependency>
178
179         <dependency>
180           <groupId>org.eclipse.jetty</groupId>
181           <artifactId>jetty-server</artifactId>
182           <version>${org.eclipse.jetty.jetty-server.version}</version>
183         </dependency>
184
185         <dependency>
186           <groupId>org.eclipse.jetty</groupId>
187           <artifactId>jetty-webapp</artifactId>
188           <version>${org.eclipse.jetty.jetty-webapp.version}</version>
189         </dependency>
```

Package: git (1:2.17.1-1ubuntu0.9

fast, scalable, distributed revision control system

Other Packages Related to git

depends · recommends · suggests · enhances

- git-man (<< 1:2.17.0~) [not amd64, i386]
  fast, scalable, distributed revision control system (manual pages)
- git-man (<< 1:2.17.1~) [amd64, i386]
- git-man (>> 1:2.17.0) [not amd64, i386]
- git-man (>> 1:2.17.1) [amd64, i386]
- libc6 (>= 2.16) [not arm64, ppc64el]
  GNU C Library: Shared libraries
  also a virtual package provided by libc6-udeb
- libc6 (>= 2.17) [arm64, ppc64el]
- libcurl3-gnutls (>= 7.16.2)
  easy-to-use client-side URL transfer library (GnuTLS flavour)
- liberror-perl
  Perl module for error/exception handling in an OO-ish way
- libexpat1 (>= 2.0.1)
  XML parsing C library - runtime library
- libpcre3
  Old Perl 5 Compatible Regular Expression Library - runtime files
- perl
  Larry Wall's Practical Extraction and Report Language
- zlib1g (>= 1:1.2.0)
  compression library - runtime
- less
  pager program similar to more
- patch
  Apply a diff file to an original
- ssh-client
  virtual package provided by openssh-client

Where are the dependencies hosted?

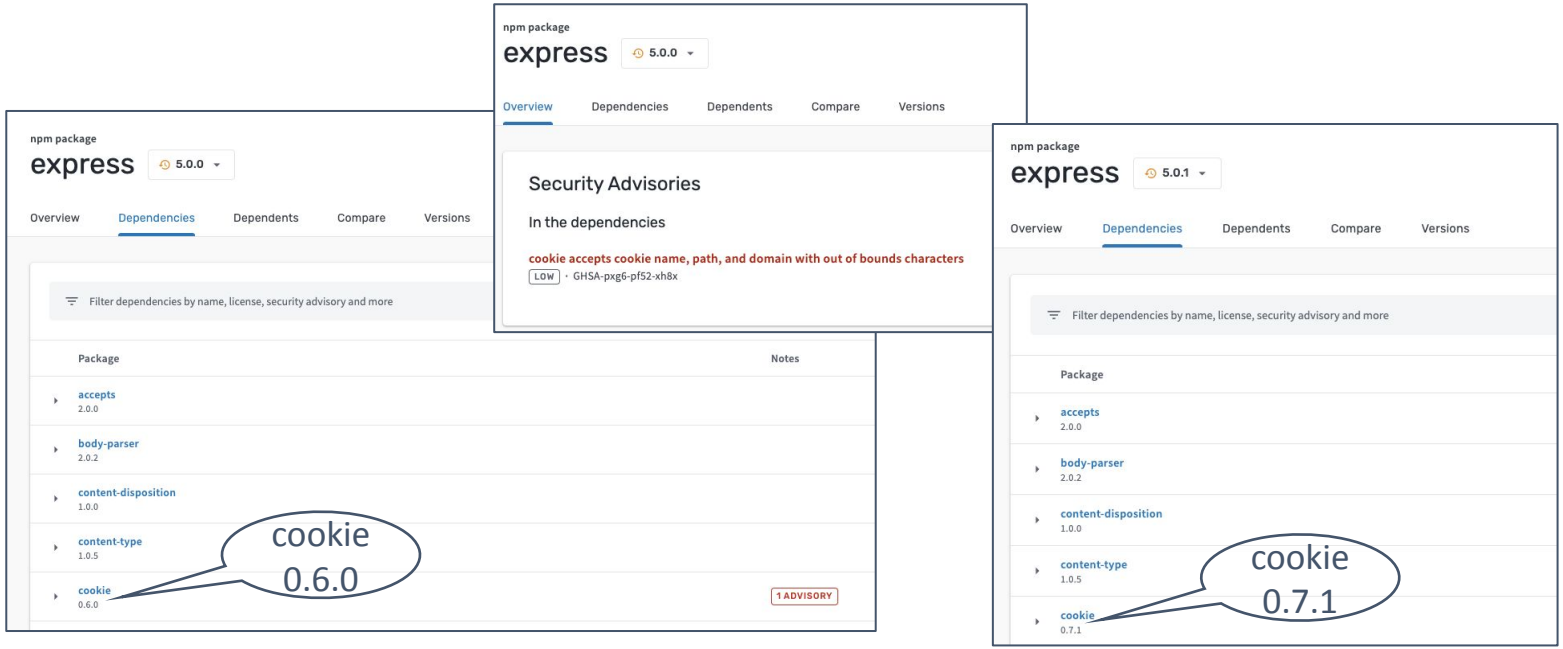
- Typical Public Repositories
  - Maven Central (Java)
  - PyPI (Python)
  - NPM Registry (JavaScript)
- Packages need a unique identifier
  - Typically a package name (sometimes owner name) and version
- Custom repositories allowed by most package managers
  - Often used for company-internal packages or cache mirroring
- Somebody needs to manage repositories
  - **Availability**: Repository needs to be running
  - **Access Control:** Packages should only be published by owners
  - **Integrity:** Packages should be signed or otherwise verifiable
  - **Uniqueness and archival:** Only one artifact per version
  - **Traceability:** Packages can have metadata pointing to source or tests
  - **Security:** ???

Dependency Pinning vs. Floating

- Pinning: ”I depend on libFoo 1.5.0”
  - Declares a specific version of the dependency. **Frozen in time.**

- Floating: “I depend on libFoo-latest”
  - Each build will pull the **latest available** libFoo version
  - (Other forms available, e.g. libFoo 1.5.x)

Pinned dependencies requires manual updates in case of security issues



Pinning vs Floating

Pinning Dependencies (e.g. 1.5.3)

✅ Reproducible builds

❌ Can become vulnerable due to dependency bugs

❌ Have to keep updating dependents as dependencies evolve

✅ Stable network effects

Floating Dependencies (e.g. 1.x)

❌ Flaky builds (breaking changes)

✅ Latest security patches & bug fixes

✅ Less manual maintenance

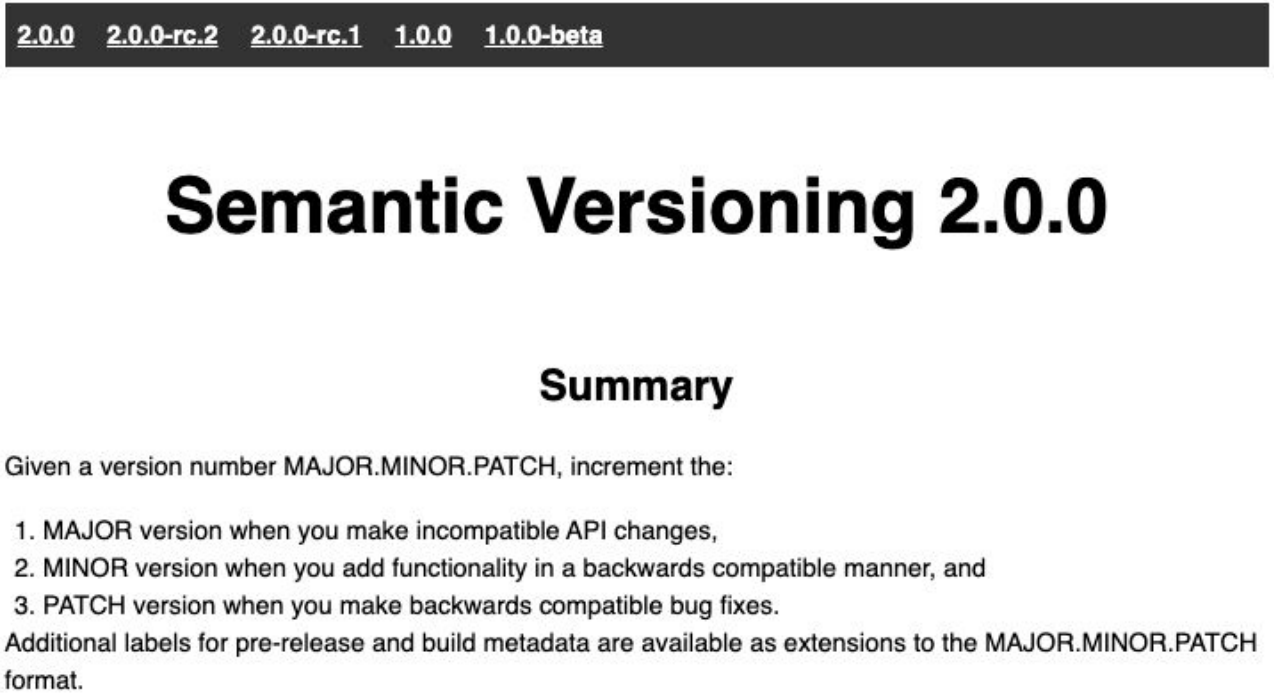
❌ Floats leak transitively

(A pin to B floating C; then A still sees changing version of C)

Semantic Versioning

- Widely used convention for versioning releases
  - E.g. 1.2.1, 3.1.0-alpha-1, 3.1.0-alpha-2, 3.1.0-beta-1, 3.1.0-rc1
- Format: {MAJOR} . {MINOR} . {PATCH}
- Each component is ordered (numerically, then lexicographically; release-aware)
  - 1.2.1 < 1.10.1
  - 3.1.0-alpha-1 < 3.1.0-alpha-2 < 3.1.0-beta-1 < 3.1.0-rc1 < 3.1.0
- Contracts:
  - MAJOR updated to indicate breaking changes
    - Same MAJOR version => backward compatibility
  - MINOR updated for additive changes
    - Same MINOR version => API compatibility (important for linking)
  - PATCH updates functionality without new API
    - Ninja edit; usually for bug fixes
- Largely dependent on honor system. No easy way to automatically verify (can you solve it?)

https://semver.org/



2.0.0 2.0.0-rc.2 2.0.0-rc.1 1.0.0 1.0.0-beta

Semantic Versioning 2.0.0

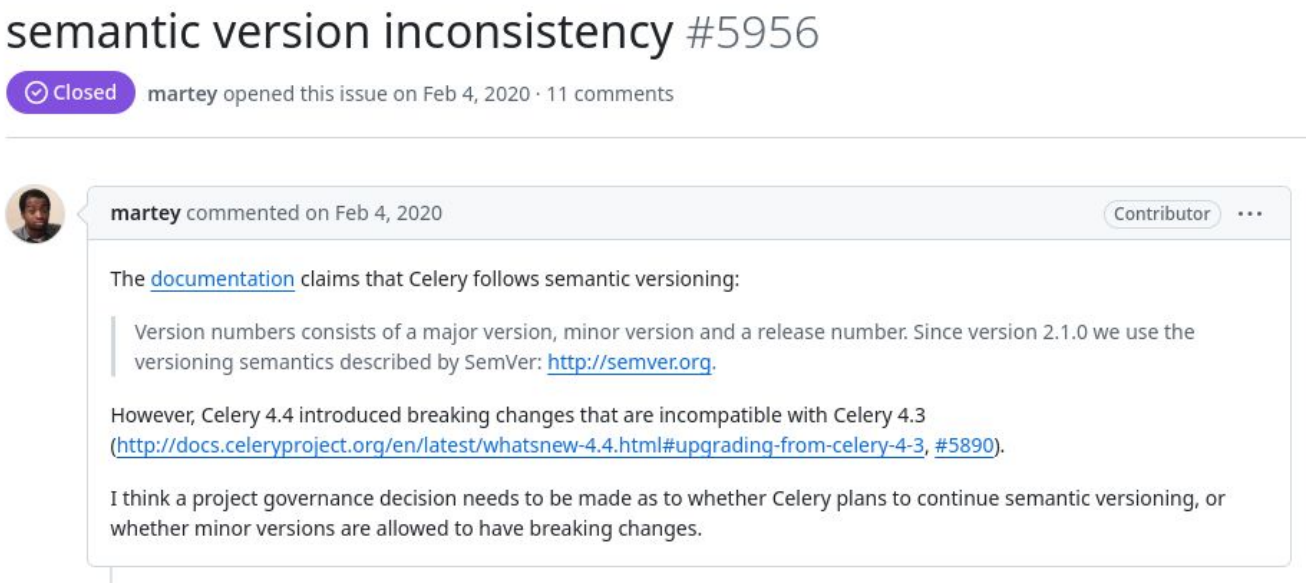
Summary

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards compatible manner, and
3. PATCH version when you make backwards compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

People rely on SemVer contracts



semantic version inconsistency #5956

Closed **martey** opened this issue on Feb 4, 2020 · 11 comments

martey commented on Feb 4, 2020 — Contributor

The documentation claims that Celery follows semantic versioning:

> Version numbers consists of a major version, minor version and a release number. Since version 2.1.0 we use the versioning semantics described by SemVer: http://semver.org.

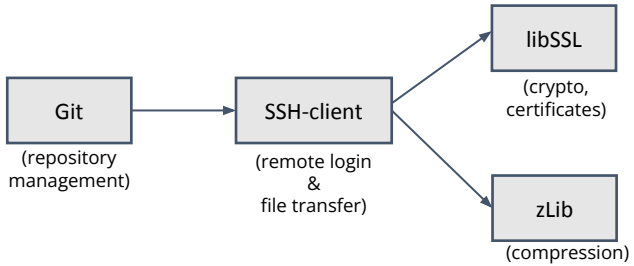
However, Celery 4.4 introduced breaking changes that are incompatible with Celery 4.3 (http://docs.celeryproject.org/en/latest/whatsnew-4.4.html#upgrading-from-celery-4-3, #5890).

I think a project governance decision needs to be made as to whether Celery plans to continue semantic versioning, or whether minor versions are allowed to have breaking changes.

See full thread for an “interesting” example of Open Source governance and communication

Transitive Dependencies

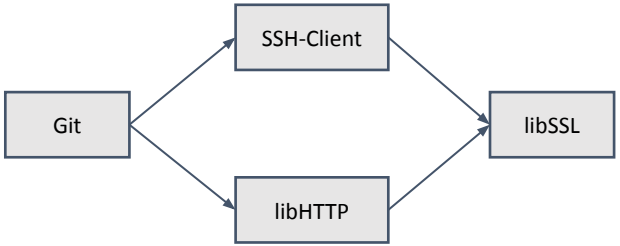
Packages can depend on other packages



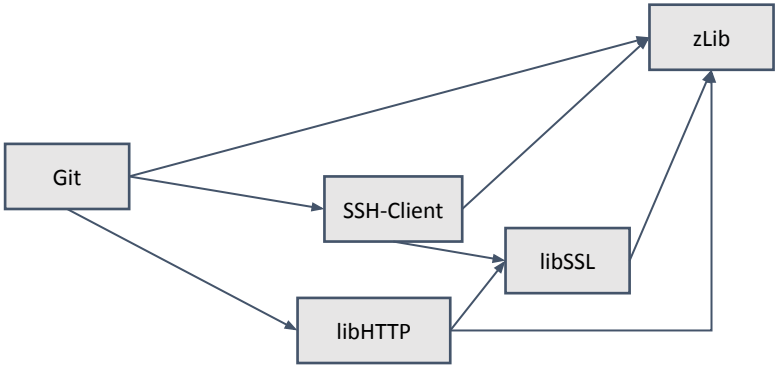
Q: Should Git be able to use exports of libSSL (e.g. certificate management) or zLib (e.g. gzip compression)?

Diamond Dependencies

What are some problems when multiple intermediate dependencies have the same transitive dependency?

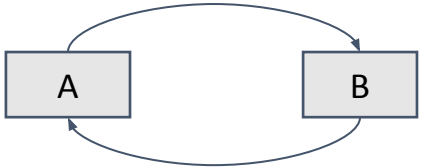


Generally, can also be across levels



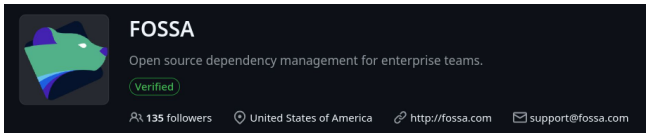
Cyclic Dependencies

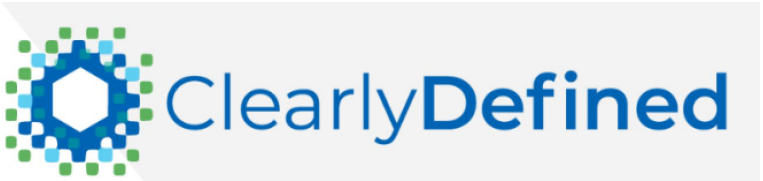
- A very bad thing
- Avoid at all costs
- Sometimes unavoidable or intentional
  - E.g. GCC is written in C (needs a C compiler)
  - E.g. Apache Maven uses the Maven build system
  - E.g. JDK tested using JUnit, which requires the JDK to compile



License Awareness from the Start

- Different open-source licenses come with different obligations.
- Always check license terms before using a dependency.
- Track licenses in your project (early, not after product launch)
  - Continuous process
- Tools:
  - FOSSA
  - ClearlyDefined
  - SPDX standards

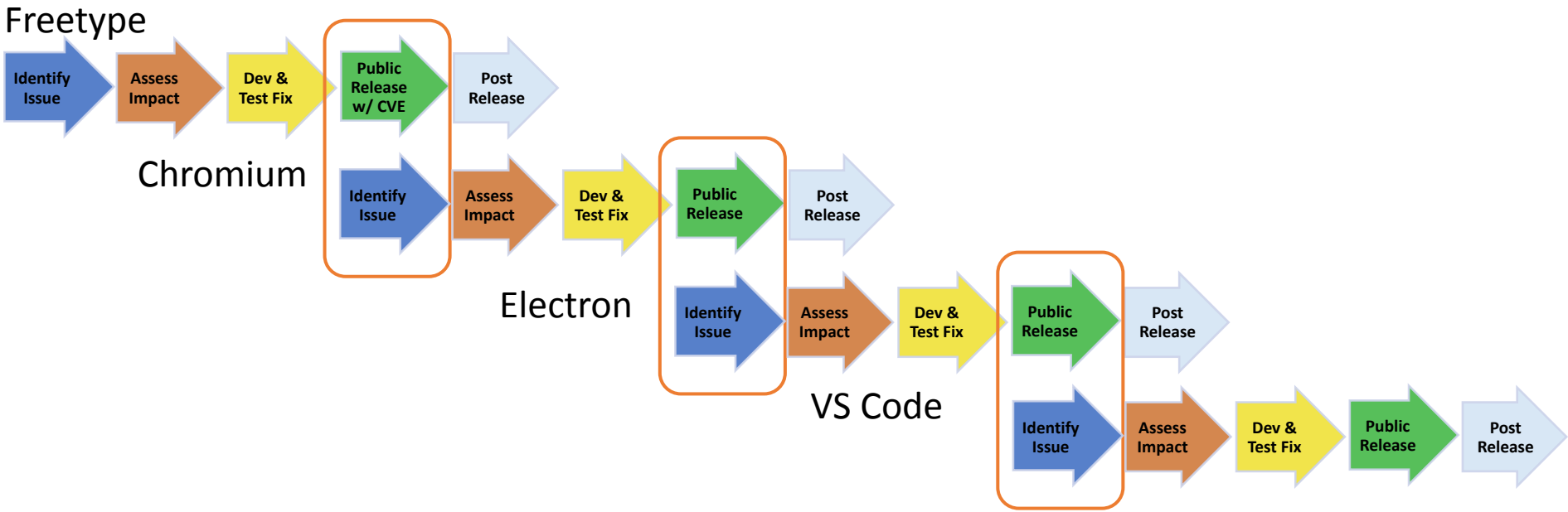





Would you eat this cookie?


Would you insert a found
drive into your laptop?

OSS Vulnerability Lifecycle



Summary: How to Manage Dependencies Well

- Regularly monitor and update dependencies.
  - Subscribe to project security advisories.
- Have a vulnerability response plan (don't wait for disasters).
- Tools SMEs could use:
  - Dependabot (GitHub automated checks for outdated packages)
  - Snyk (open source vulnerability scanning)
  - OWASP Dependency-Check
- Track the licenses of all third-party code you use.
- Maintain a Software Bill of Materials (SBOM) listing all dependencies.

Open-source components are powerful, but managing them carefully is essential for security, compliance, and stability.
Good dependency management protects your project and speeds up your team's work.