

Open-Source Software

17-313 Fall 2023

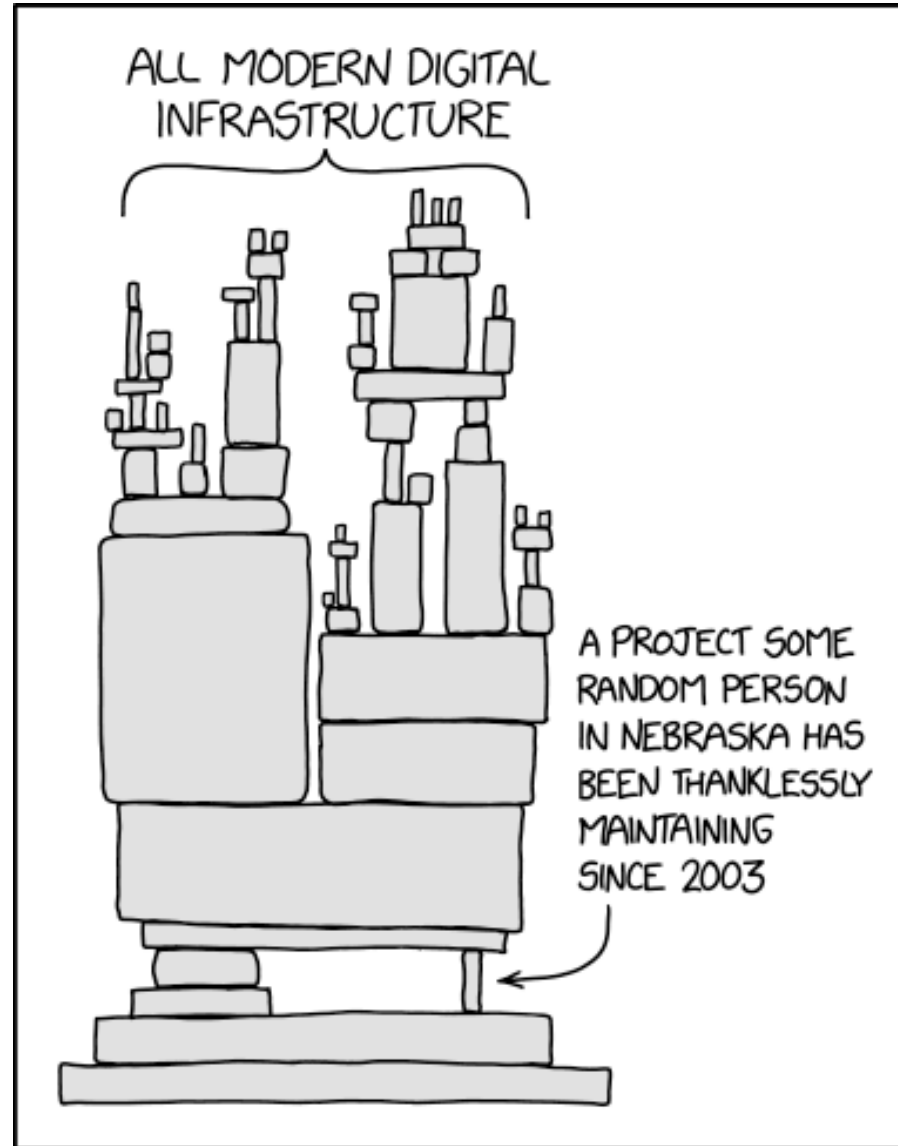
Foundations of Software Engineering

<https://cmu-313.github.io>

Andrew Begel and Rohan Padhye

Learning Goals

- Distinguish between open-source software, free software, and commercial software.
- Identify the common types of software licenses and their implications.
- Distinguish between copyright and intellectual property.
- Express an educated opinion on the philosophical/political debate between open source and proprietary principles.
- Describe how open-source ecosystems work and evolve, in terms of maintainers, community contribution, and commercial backing
- Identify various concerns of commercial entities in leveraging open-source, as well as strategies to mitigate these.

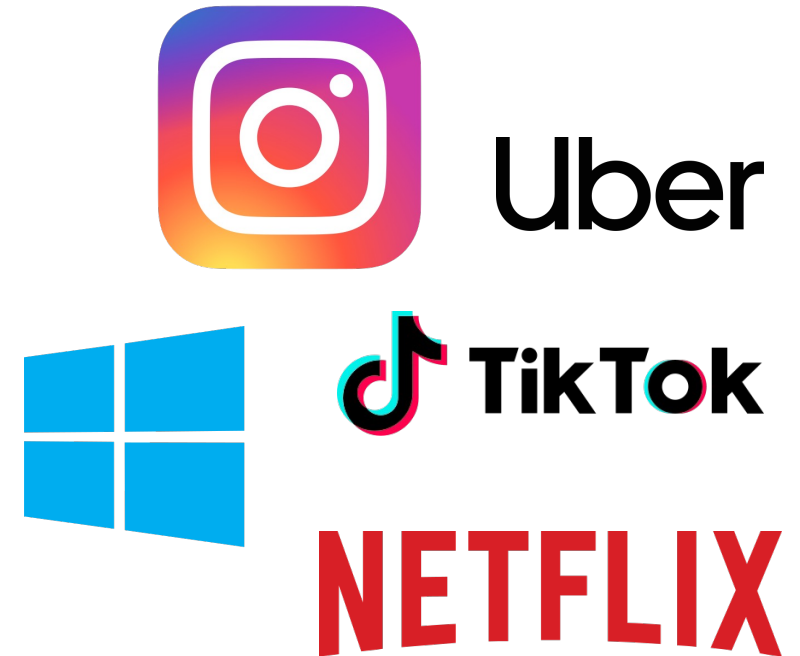


What is Open-Source Software?

Open-source



Proprietary



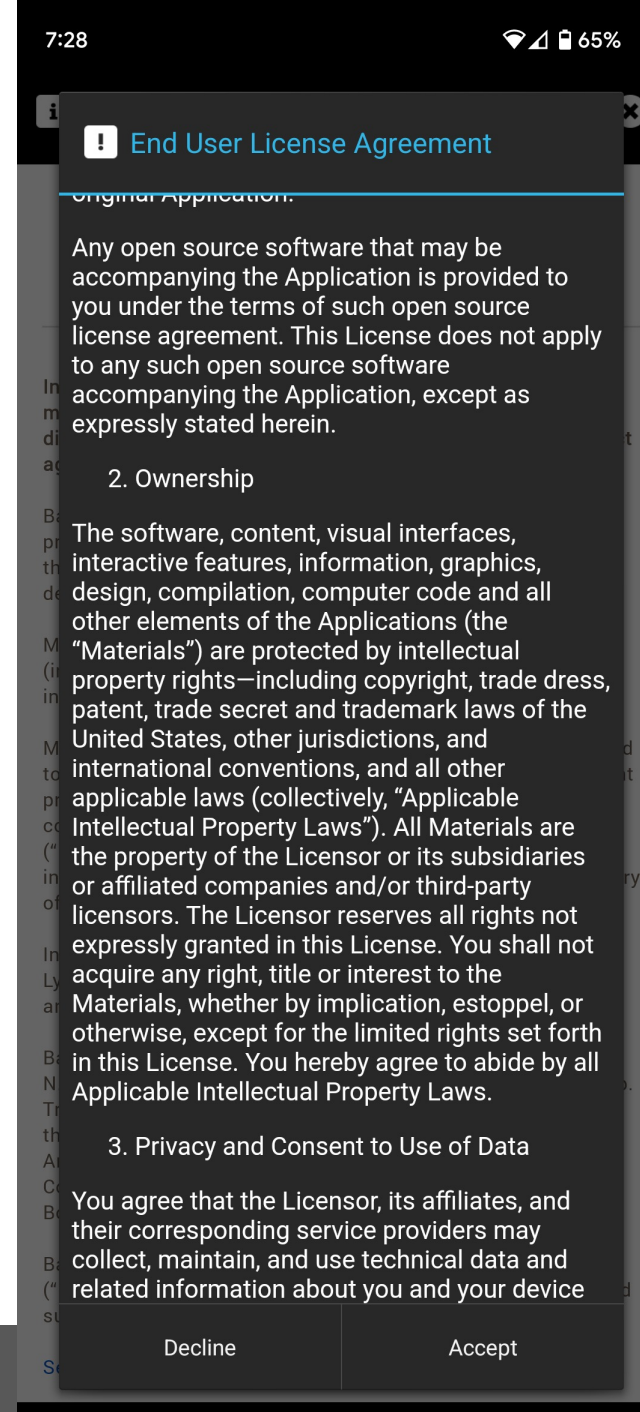
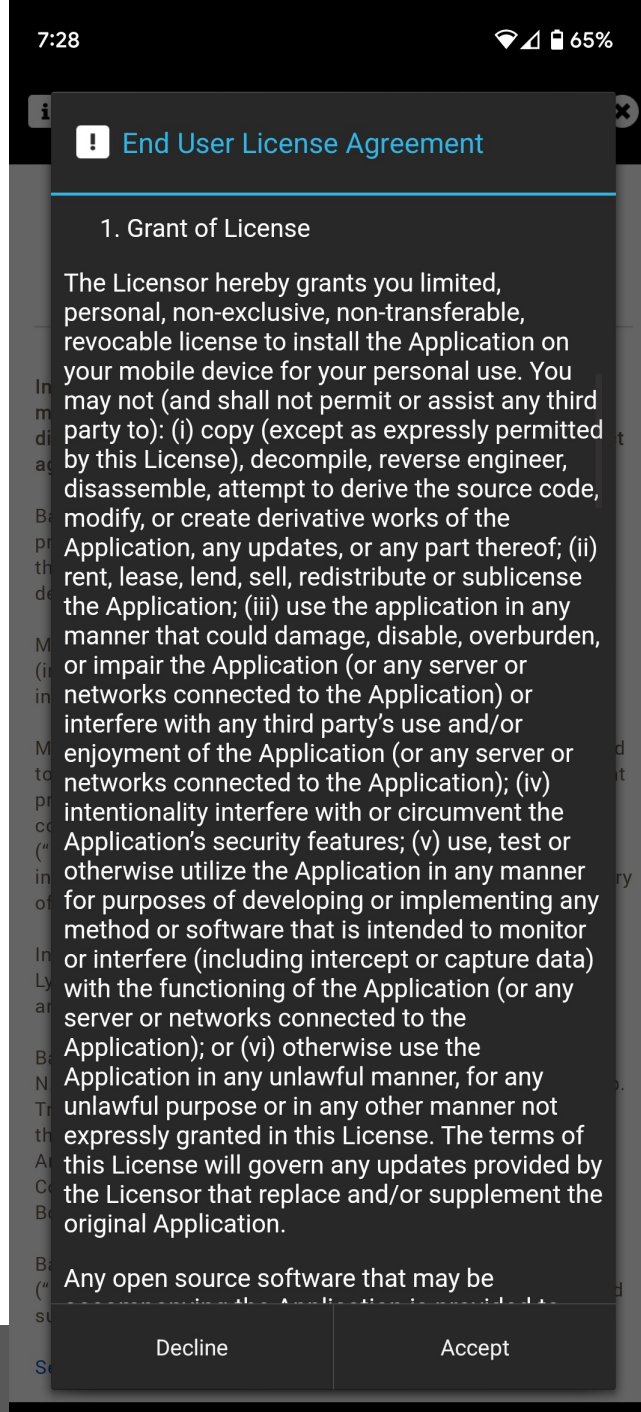
What is Open-Source Software (OSS)?

- Source code availability
- Right to modify and create derivative works
- (Often) Right to redistribute derivative works

Contrast with proprietary software: a black box

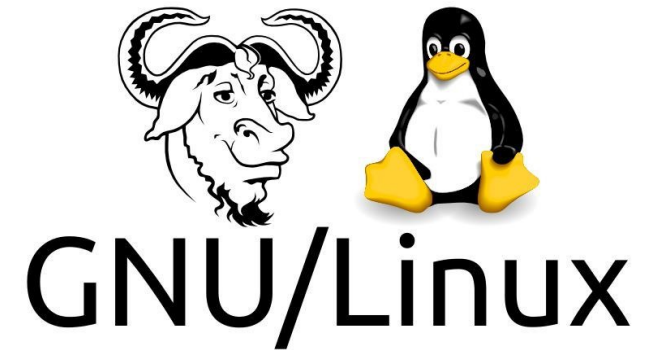
- Intention is to be used, not examined, inspected, or modified.
- No source code – only download a binary (e.g., an app) or use via the internet (e.g., a web service).
- Often contains an End User License Agreement (EULA) governing rights and liabilities.
- EULAs may specifically prohibit attempts to understand application internals.

Example: Bank
app on my phone



Free Software vs. Open Source

- Free software origins (70-80s ~Stallman)
 - ~~Cultish~~ Political goal
 - Software part of free speech
 - free exchange, free modification
 - proprietary software is unethical
 - security, trust
 - GNU project, Linux, GPL license
- Open source (1998 ~O'Reilly)
 - Rebranding without political legacy
 - Emphasis on internet and large dev/user involvement
 - Openness toward proprietary software/coexist
 - (Think: Netscape becoming Mozilla)





Perception (from some):

- Anarchy
- Demagoguery
- Ideology
- Altruism

Why Go Open Source (vs. Proprietary) ?

Advantages

- <today's activity; do in groups>

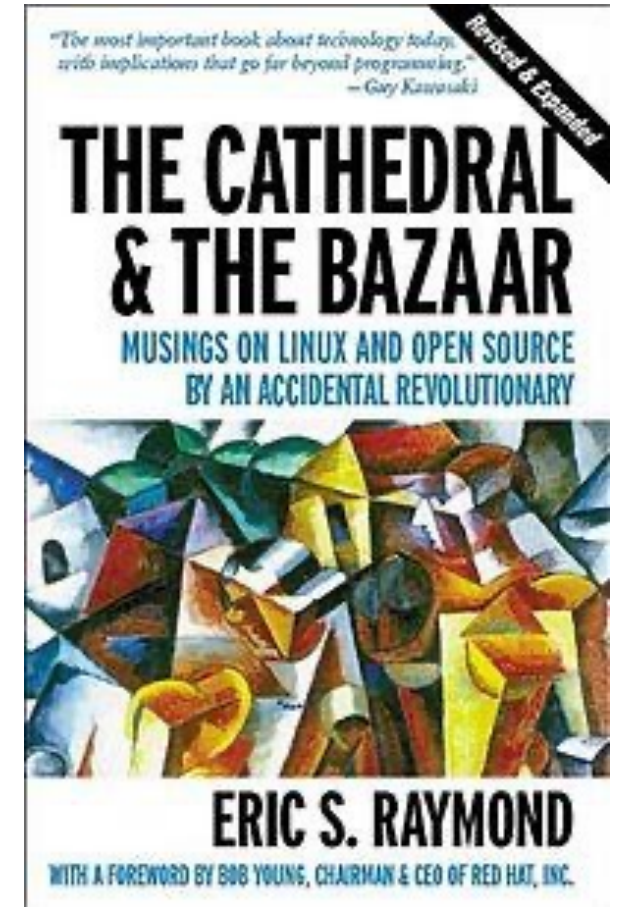
Disadvantages

- <make sure to note down names of people sitting next to you>

Open-Source Ecosystems

How OSS is developed

The Cathedral and the Bazaar



The Bazaar won

Cathedral

- Developed centrally by a core group of members
- Available for all once complete (or at releases)
- Examples: GNU Emacs, GCC (back in the 1990s)
- “Sort-of” examples today: Chrome, IntelliJ

Bazaar

- Developed openly and organically
- Wide participation (in theory, anyone can contribute)
- Examples: Linux

OSS has many stakeholders / contributors

- Core members
 - Often (but not always) includes the original creators
 - Direct push access to main repository
 - May be further split into admin roles and developers
- External contributors
 - File bug reports and report other issues
 - Contribute code and documentation via pull requests
- Other supporters
 - Beta testers (users)
 - Sponsors (financial or platform)
 - Steering committees or public commenters (for standards and RFCs)
- Spin-offs
 - Maintainers of forks of the original repository

Contributing processes

- Mature OSS projects often have strict contribution guidelines
 - Look for CONTRIBUTING.md or similar
- Common requirements:
 - Coding style (recall: linters) and passing static checks
 - Inclusion of test cases with new code
 - Minimum number of code reviews from core devs
 - Standards for documentation
 - Contributing licensing agreements (more on that later)

Governance

- Some OSS projects are managed by for-profit firms
 - **Examples:** Chromium (Google), Moby (Docker), Ubuntu (Canonical), TensorFlow (Google), PyTorch (Meta), Java (Oracle)
 - Contributors may be a mix of employees and community volunteers
 - Corporations often fund platforms (websites, test servers, deployments, repository hosting, etc.)
 - Corporations usually control long-term vision and feature roadmap
- Many OSS projects are managed by non-profit foundations or ad-hoc communities
 - **Examples:** Apache Hadoop/Spark/Hbase/Kafka/Tomcat (ASF), Firefox (Mozilla), Python (PSF), NumPy (community)
 - Foundations fund project infrastructure via charitable donations
 - Long-term vision often developed via a collaborative process (e.g., Apache) or by benevolent dictators (e.g., Python, Linux)
- Corporations still heavily rely on community-owned OSS projects
 - Many OSS non-profits are funded by Big Tech (e.g., Mozilla by Google)

Example: Apache

WHAT MAKES THE APACHE WAY SO HARD TO DEFINE?

The Apache Way is a living, breathing interpretation of one's experience with our community-led development process. Apache is unique, diverse, and focused on the activities needed at a particular stage of the project's lifetime, including nurturing community building awareness. What is important is that they embrace:

- **Earned Authority:** all individuals are given the opportunity to participate, but their influence is based on publicly earned community. Merit lies with the individual, does not expire, is not influenced by employment status or employer, and is not project cannot be applied to another). [More on merit.](#)
- **Community of Peers:** individuals participate at the ASF, not organizations. The ASF's flat structure dictates that roles are equal weight, and contributions are made on a volunteer basis (even if paid to work on Apache code). The Apache community with respect in adherence to our [Code of Conduct](#). Domain expertise is appreciated; Benevolent Dictators For Life are disallowed. [participation.](#)
- **Open Communications:** as a virtual organization, the ASF requires all communications related to code and decision-making to be asynchronous collaboration, as necessitated by a globally-distributed community. Project mailing lists are archived, public, and searchable.
 - dev@ (primary project development)
 - user@ (user community discussion and peer support)
 - commits@ (automated source change notifications)
 - occasionally supporting roles such as marketing@ (project visibility)

...as well as restricted, day-to-day operational lists for Project Management Committees. Private decisions on code, policies, or project direction and transactions must be brought on-list. More on [communications](#) and the [use of mailing lists](#).

- **Consensus Decision Making:** Apache Projects are overseen by a self-selected team of active volunteers who are contributing to the project. Projects are auto-governing with a heavy slant towards driving consensus to maintain momentum and productivity. While consensus is the goal, establish at all times, holding a vote or other coordination may be required to help remove any blocks with binding decisions. [More on decision making and voting.](#)
- **Responsible Oversight:** The ASF governance model is based on trust and delegated oversight. Rather than detailed rules, governance is principles-based, with self-governing projects providing reports directly to the Board. Apache Committers review commits, employing mandatory security measures, ensuring license compliance, and protecting the Apache community from abuse. [More on responsibility.](#)



OUR SPONSORS

The Apache Software Foundation could not exist without the continued generous support from the community. We would like to take this opportunity to thank our sponsors. If you are interested in sponsoring the ASF, please read our [sponsorship page](#).

FOUNDATION SPONSORS

Platinum Sponsors:

FACEBOOK yahoo!

Facebook

Yahoo!

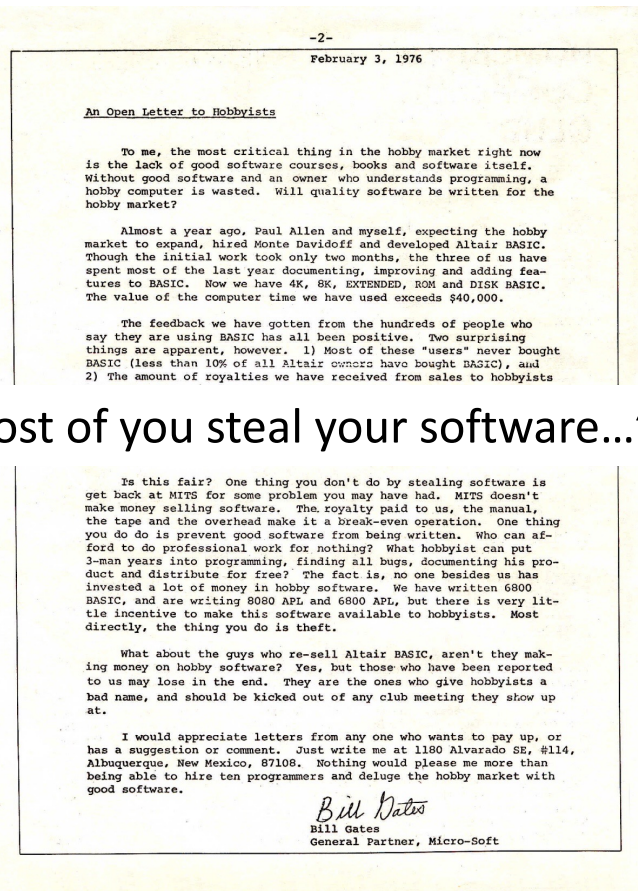


Apple



Google

Corporate outlook towards open-source has evolved over the years



“...most of you steal your software...”

Redmond top man Satya Nadella: 'Microsoft LOVES Linux'

Open-source 'love' fairly runneth over at cloud event



20 Oct 2014 at 23:45, Neil McAllister



Risks in *not* open-sourcing?

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling ma-

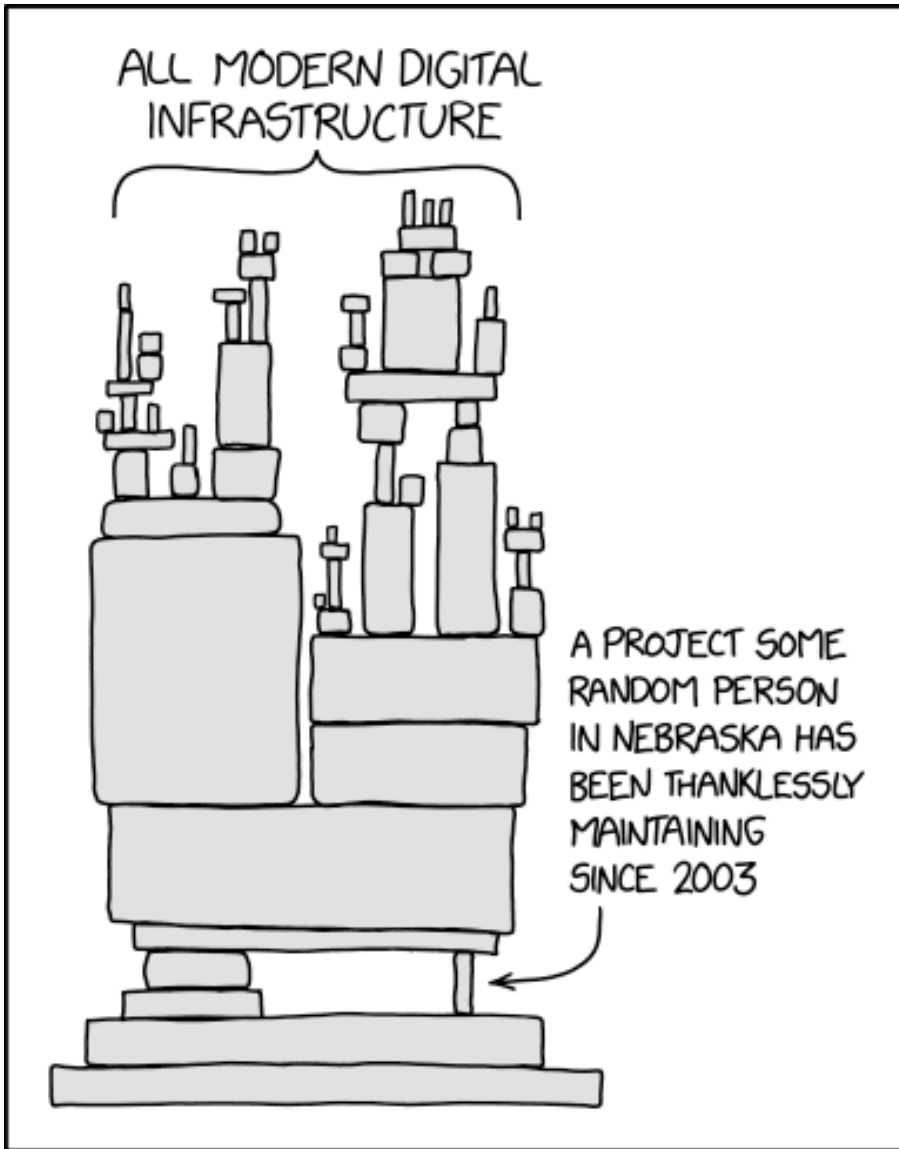
given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp



Use of open source software within companies

- Is the license compatible with our intended use?
 - More on this later
- How will we handle versioning and updates?
 - Does every internal project declare its own versioned dependency or do we all agree on using one fixed (e.g., latest) version?
 - Sometimes resolved by assigning internal “owners” of a third-party dependency, who are responsible for testing updates and declaring allowable versions.
- How to handle customization of the OSS software?
 - Internal forks are useful but hard to sync with upstream changes.
 - One option: Assign an internal owner who keeps internal fork up-to-date with upstream.
 - Another option: Contribute all customizations back to upstream to maintain clean dependencies.
- Security risks? Supply chain attacks on the rise.



QUARTZ | Make business better.™

HOME LATEST BUSINESS NEWS MONEY & MARKETS TECH & INNOVATION LIFESTYLE LEADERSHIP EMAILS PODCASTS EN ESPAÑOL

We may earn a commission from links on this page.

TECH & INNOVATION

NPM ERR!

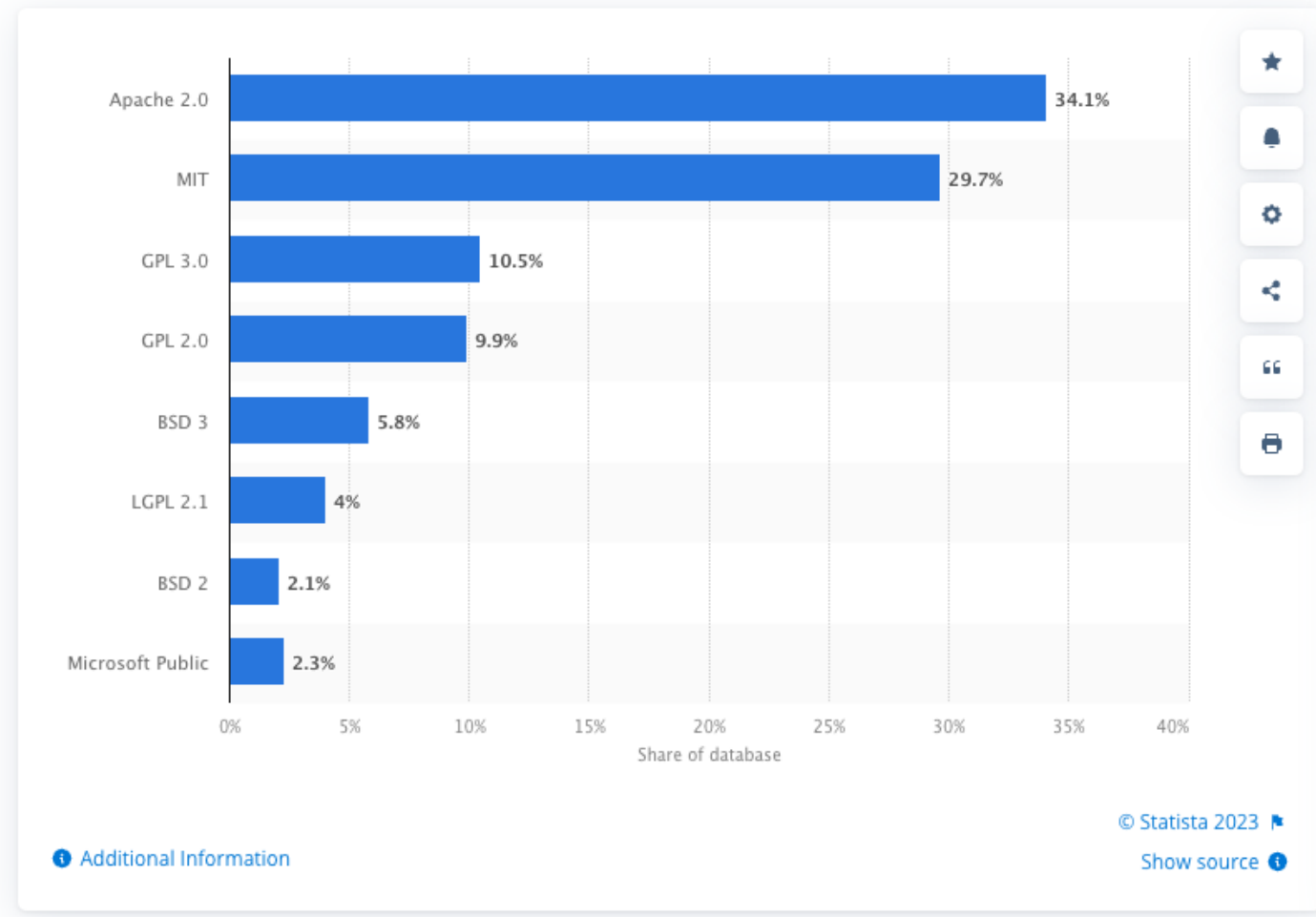
How one programmer broke the internet by deleting a tiny piece of code

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

Software Licenses

Note: I am not a lawyer (this is not legal advice)

Most popular open source licenses worldwide in 2021




Which license to choose?

← → ↻ 🏠 choosealicense.com

Choose an open source license

An open source license protects contributors and users. Businesses and savvy developers won't touch a project without this protection.


{ Which of the following best describes your situation? }



I need to work in a community.

Use the [license preferred by the community](#) you're contributing to or depending on. Your project will fit right in.


If you have a dependency that doesn't have a license, ask its maintainers to [add a license](#).



I want it simple and permissive.

The [MIT License](#) is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

[Babel](#), [.NET](#), and [Rails](#) use the MIT License.



I care about sharing improvements.

The [GNU GPLv3](#) also lets people do almost anything they want with your project, *except* distributing closed source versions.

[Ansible](#), [Bash](#), and [GIMP](#) use the GNU GPLv3.

{ What if none of these work for me? }

My project isn't software.

[There are licenses for that.](#)

I want more choices.

[More licenses are available.](#)

I don't want to choose a license.

[Here's what happens if you don't.](#)

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 relicensed under the same license (copyleft)

Risks of “copyleft” licenses

- Example: GNU GPL
- Require licensing derivative works also with same license
 - This is intentional!
- Depending on a GPL project from within a proprietary or differently-licensed codebase is disaster
 - Viral effect of polluting everything else with GPL requirement
- Most companies will avoid GPL code with a ten-foot pole
 - Expect vetting process before engineers are allowed to use third-party libraries from GitHub, etc.

Lesser GNU Public License (LGPL)

- Software must be a library
- Similar to GPL but does not consider dynamic binding as “derivative work”
- So, proprietary code can depend on LGPL libraries as long as they are not being modified
- See also: GPL with classpath exception (e.g., Oracle JDK)

MIT License

- Simple, commercial-friendly license
- Must retain copyright credit
- Software is provided as is
- Authors are not liable for software
- No other restrictions

Apache License

- Similar to MIT license
- 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

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

Creative Commons (CC)

- More common for licensing data-sets instead of code
 - Examples: images, websites, documentation, slides, plots, videos
- CC-BY (attribution only; derivatives allowed)
- CC-BY-SA (attribution and share-alike for derivatives)
- CC-BY-ND (attribution and no derivatives)

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

Copyright vs. Intellectual Property (IP)

- IP and Patents cover an idea for solving a problem
 - Examples: Machine designs, pharma processes to manufacture certain drugs, (controversially) algorithms
 - Have expiry dates. IP can be licensed or sold/transferred for \$\$\$.
- Copyrights cover particular expressions of some work
 - Examples: Books, music, art, source code
 - Automatic copyright assignment to all new work unless a license authorizes alternative uses.
- Exceptions for trivial works and ideas.

Contributor Licensing Agreements (CLA)

- Often a requirement to sign these before you can contribute to OSS projects
 - Scoped only to that project
- Assigns the maintainers specific rights over code that you contribute
 - Without this, you own the copyright and IP for even small bug fixes and that can cause them legal headaches in the future

Summary

- Open-source software harnesses the collective power of stakeholders not directly associated with main developers
- Open-source ecosystems thrive in many application domains where reuse is common (e.g., platforms, frameworks, libraries)
- Corporations rely on open-source even if they develop proprietary software or services.
- Open-source licenses must be chosen carefully to align with intended use case.
- You will all contribute to OSS in this class!