Open Source

17-313: Foundations of Software Engineering

https://cmu-313.github.io

Michael Hilton and Josh Sunshine

Spring 2026

Administrivia

Project 4B - Final Deliverables - Due Friday, April 3, 2026 @ 11:59PM

Project 5 Published

No class Thursday April 9, Carnival

Midterm 2 - Thursday April 16

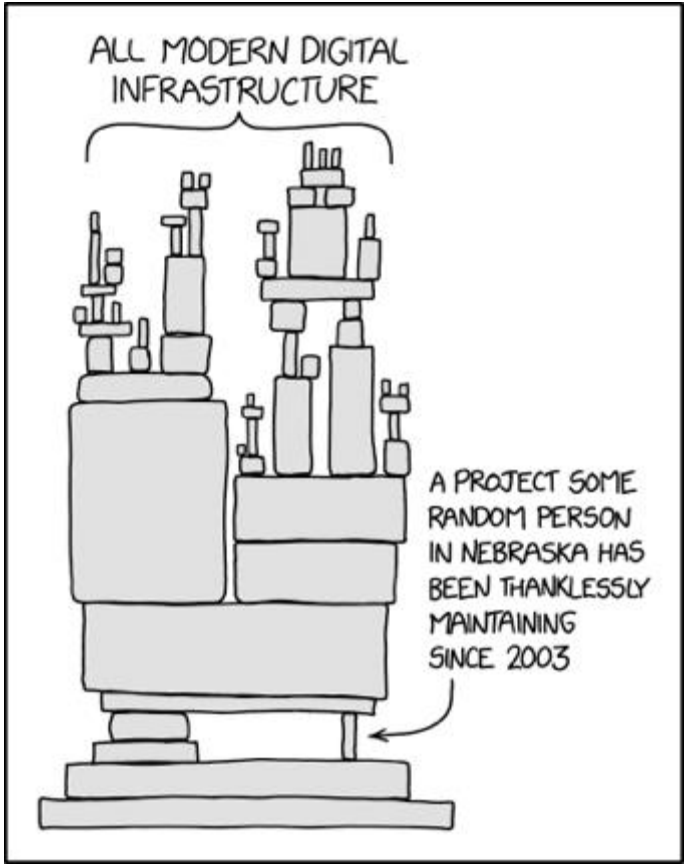
Learning Goals

- 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.

Open Source

Background: laws and open source

- Copyright protects creative, intellectual and artistic works — including software
- Alternative: public domain (nobody may claim exclusive property rights)
- Trademark protects the name and logo of a product
- OSS is generally copyrighted, with copyright retained by contributors or assigned to entity that maintains it
- Copyright holder can grant a license for use, placing restrictions on how it can be used (perhaps for a fee)


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

What is Open-Source Software?

Open-source

Proprietary

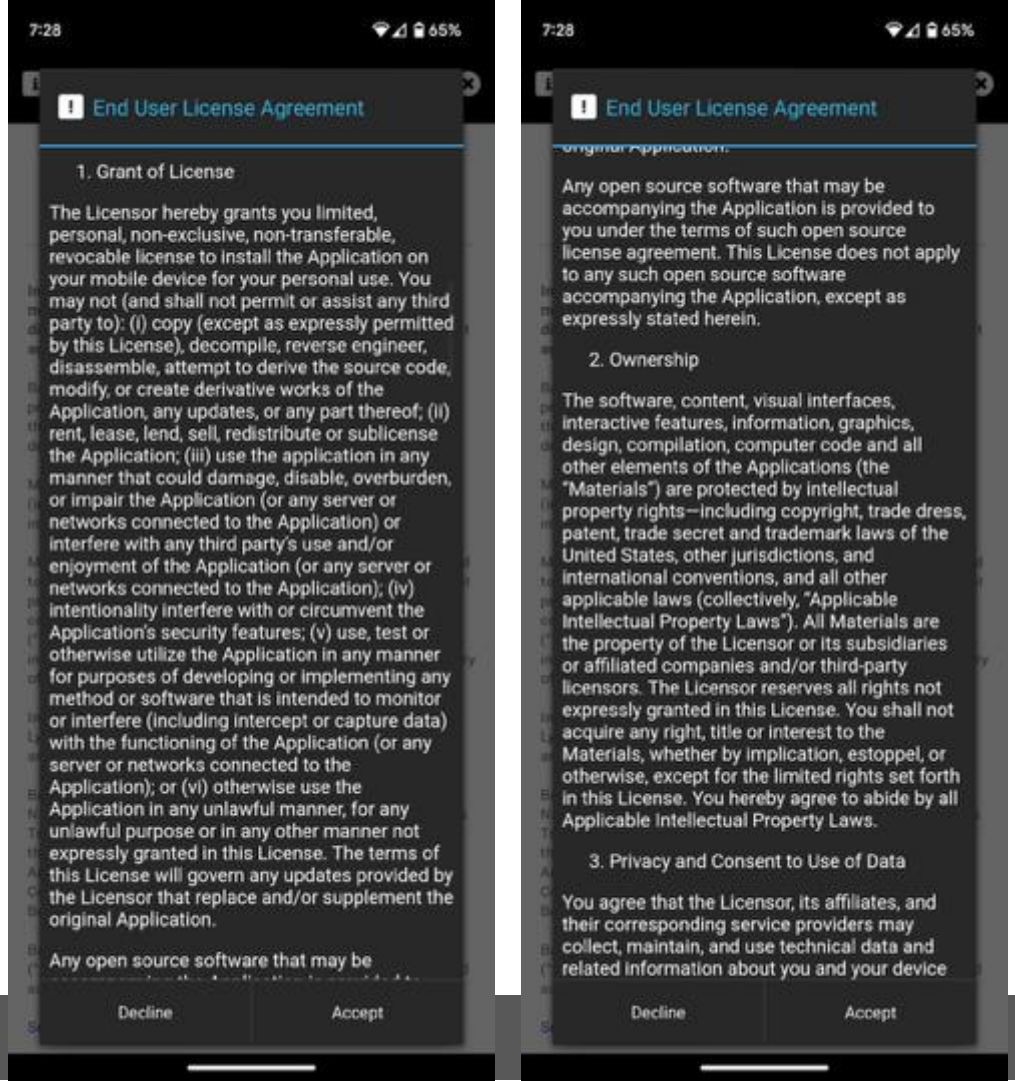
What is Open-Source Software (OSS)?

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



Early open source: UNIX to BSD

- Hardware was not yet standardized, computer vendors focused on hardware, building new operating systems for each platform
- Much software development focused in academic labs, and AT&T's Bell Labs
- Unix created at Bell Labs using the new, portable language "C", licenses initially released with source code
- 1978: UC Berkeley begins distributing their own derived version of Unix (BSD)
- AT&T is prohibited from entering new telecommunications businesses (can't make OS a product)

IBM 704 at NASA Langley in 1957 (Public domain)

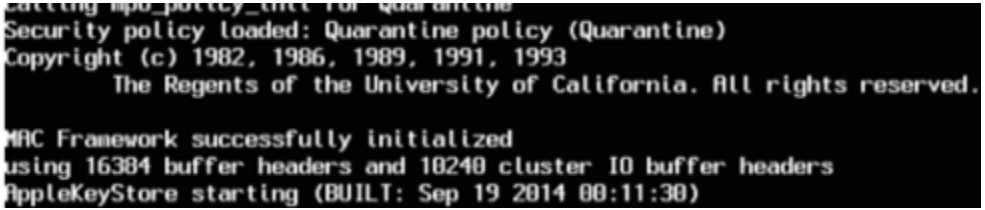
The BSD License is Permissive

- Authors of BSD created a license for the OS that:

1. Required those using it to credit the university
2. Limited liability for (mis)-use

Copyright (c) <year>, <copyright holder> All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
1.Redistributions of **source code must retain the above copyright notice, this list of conditions and the following disclaimer**.
2.Redistributions in **binary form must reproduce the above copyright notice, this list of conditions** and the following disclaimer in the documentation and/or other materials provided with the distribution.
3.All **advertising materials mentioning features** or use of this software must display the following acknowledgement: This product includes software developed by the <copyright holder>.
4.Neither the name of the <copyright holder> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED BY <COPYRIGHT HOLDER> *AS IS* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED …. (move waivers of liability)

Original BSD license



UNIX to GNU's Not Unix

- Timeline
  - 1978: UC Berkeley begins distributing their own derived version of Unix (BSD)
  - 1983: AT&T broken up by DOJ, UNIX licensing changed: no more source releases
  - Competing commercial vendors all package and sell their derivations of UNIX (AT&T, HP, Sun, IBM, SGI)
  - Also 1983: "Starting this Thanksgiving I am going to write a complete Unix-compatible software system called GNU (Gnu's Not Unix), and give it away free to everyone who can use it"

GNU logo (a gnu wildebeest)

Free software as a Philosophy

- “Free as in Speech, not as in beer”

Richard Stallman’s Free Software Foundation — free as in liberties

- Freedom 0: run code as you wish, for any purpose
- Freedom 1: study how code works, and change it as you wish
- Freedom 2: redistributed copies (of original) so you can help others
- Freedom 3: distribute copies of your modified version to others

Richard M Stallman (Licensed under GFDL)

Free software as a Philosophy

- “Free as in Speech, not as in beer”

FSF: software licensed under GNU Public License (GPL), considering questions like:

  - Required to redistribute modifications (under same license)? Yes, “copyleft"
  - Can you combine it with more restrictive licenses? No, not even with BSD!

Alternative (more like BSD):

“Do whatever you want with this software, but don’t blame me if it doesn’t work” freeware

Copyleft v. permissive

- Can I combine OSS with my product, releasing my product under a different license (perhaps not even OS)?
- Permissive licenses encourage adoption by permitting this practice
- Copyleft "protects the commons" by having all linked code under same license, transitively requiring more sharing

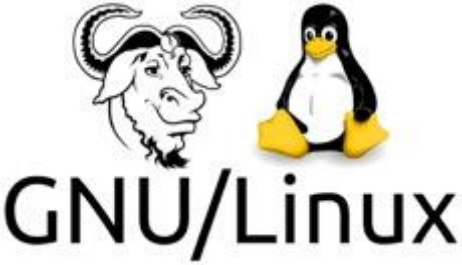
- Philosophy: do we force participation, or try to grow/incentivize it in other ways?

GNU/Linux (1991-Today)

- Stallman set out to build an operating system in 1983, ended up building utilities needed by an operating system (compiler, etc)
- Linux is built around and with the GNU utilities, licensed under GPL
- Rise of the internet, demand for internet servers drives demand for cheap/free OS
- Companies adopted and support Linux for enterprise customers
- IBM committed over $1B; Red Hat and others

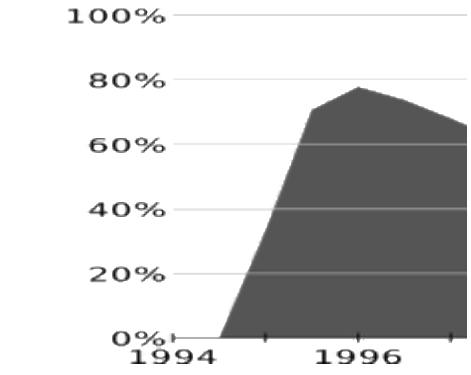
Free Software vs. Open Source

- 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)



Netscape's open source gambit

- Netscape was dominant web browser early 90's
- Business model: free for home and education use, companies pay
- Microsoft entered browser market with Internet Explorer, bundled with Windows95, soon overtakes Netscape in usage (free with Windows)
- January 1998: Netscape first company to open source code for proprietary product (Mozilla)




ZD NET

Home / Business / Enterprise Software

Netscape unveils its Navigator source code site

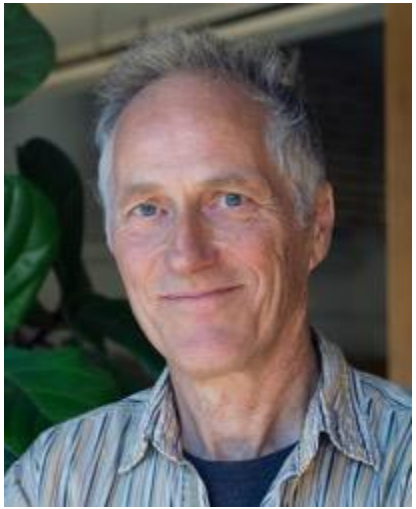
Netscape Communications Corp. is rallying its troops for next month's release of the source code for the company's Navigator Web browser.

Written by Maria Seminerio, Contributor on Feb. 22, 1998

Netscape creates a new license and model

- Until Netscape, much of OSS was the FSF and its GPL
- Open Source coined in 1998 by the Open Source Initiative to capture Netscape's aim for an open development process
- New licenses follow, e.g. MIT, Apache, etc. just like BSD, but without the advertising part
- Publisher Tim O'Reilly organizes a Freeware Summit later in 1998, soon rebranded as Open Source Summit
- Open Source is a development methodology; free software is a social movement
  — Richard Stallman

Open source initiative logo



Tim O'Reilly
Photo via Christopher Michel/Flickr, CC BY 2.0

Perception (from some):

- Anarchy
- Demagoguery
- Ideology
- Altruism

Why Go Open Source (vs. Proprietary) ?

Advantages

Disadvantages

Why Go Open Source (vs. Proprietary) ?

Advantages

- Transparency, gain user trust
- Many eyes: crowd-source bug reports and fixes
- Security: more likely for vulnerabilities to be quickly identified
- Community and adoption: get others to contribute features, build stuff around you, or fork your project

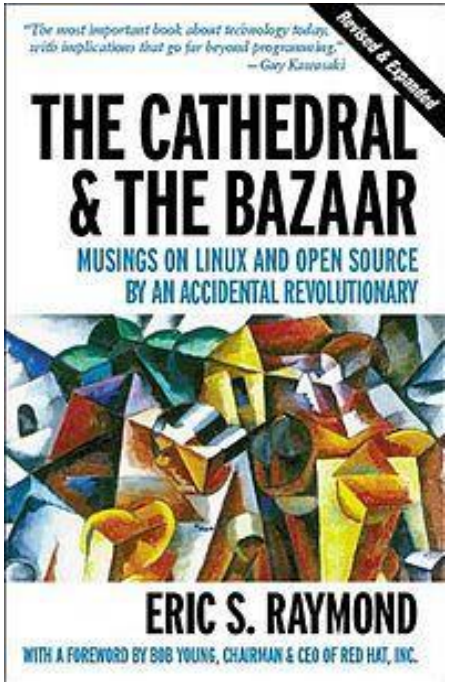
Disadvantages

- Reveal implementation secrets
- Many eyes: users can find faults more easily
- Security: more likely for others to find vulnerabilities first
- Control: You may not be able to influence the long-term direction of your platform

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)

Governence

- 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?

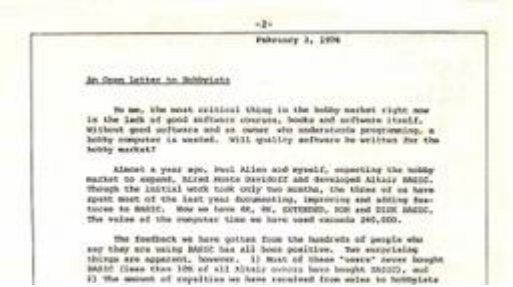
The Apache Way is a living, breathing interpretation of one's experience with our community-led development process. Apach
unique, diverse, and focused on the activities needed at a particular stage of the project's lifetime, including nurturing comm
building awareness. What is important is that they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

- ***Earned Authority:*** all individuals are given the opportunity to participate, but their influence is based on publicly earnec
community. Merit lies with the individual, does not expire, is not influenced by employment status or employer, and is n
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 ar
equal weight, and contributions are made on a volunteer basis (even if paid to work on Apache code). The Apache comr
with respect in adherence to our Code of Conduct. Domain expertise is appreciated; Benevolent Dictators For Life are d
participation.
- ***Open Communications:*** as a virtual organization, the ASF requires all communications related to code and decision-ma
asynchronous collaboration, as necessitated by a globally-distributed community. Project mailing lists are archived, pub
  - 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
discourse 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 contrib
Projects are auto-governing with a heavy slant towards driving consensus to maintain momentum and productivity. WH
establish at all times, holding a vote or other coordination may be required to help remove any blocks with binding deci
More on decision making and voting.
- ***Responsible Oversight:*** The ASF governance model is based on trust and delegated oversight. Rather than detailed rul
governance is principles-based, with self-governing projects providing reports directly to the Board. Apache Committers
reviewed commits, employing mandatory security measures, ensuring license compliance, and protecting the Apache
abuse. More on responsibility.

Corporate outlook towards open-source has evolved over the years



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 useless. 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

"…most of you steal your software…"

In this field? 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 distributing 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

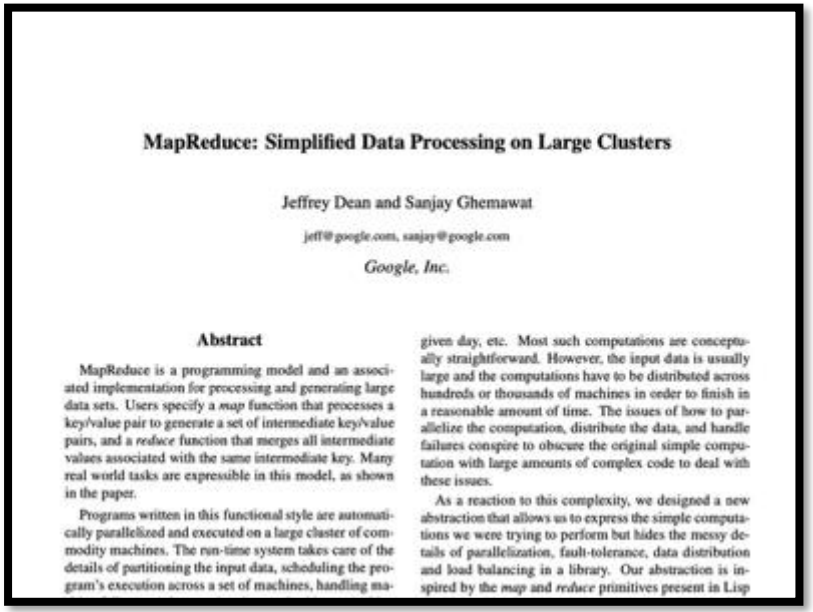
Bill Gates
General Partner, Micro-Soft

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 machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

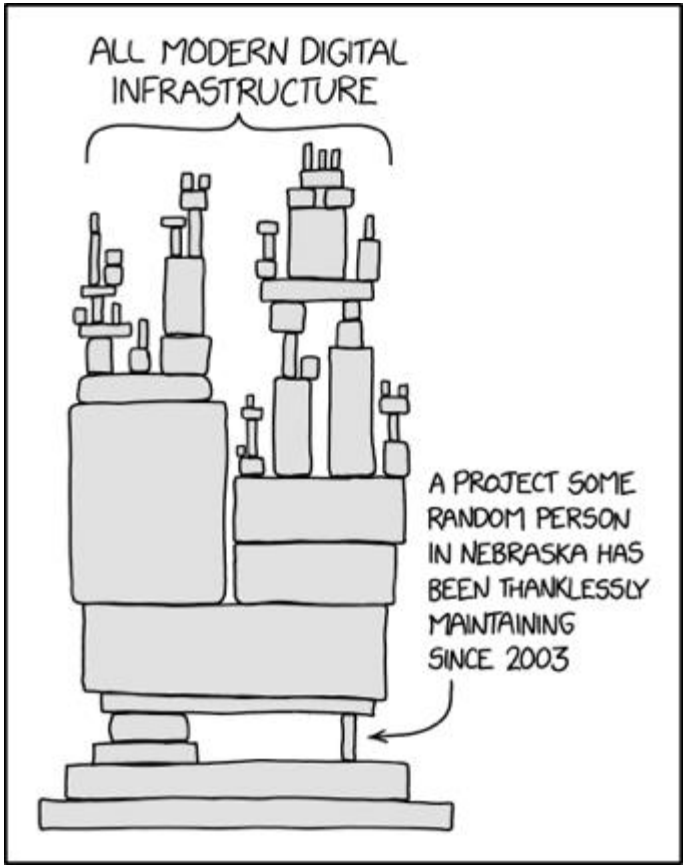
1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a 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.™ Send us a Tip! MEMBERSHIP

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

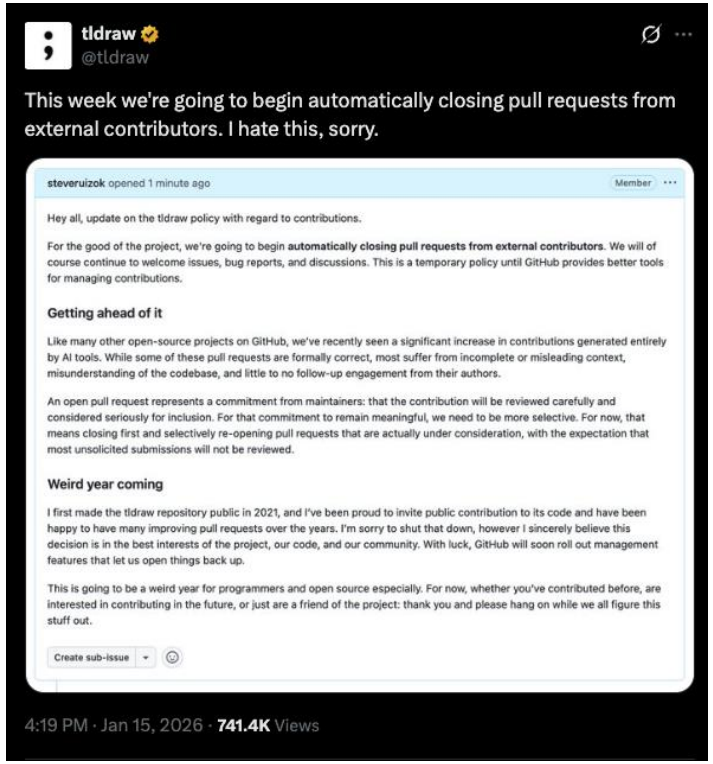
NPM ERR!

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
```

“AI is burning out the people who keep open source alive”

“Research on open source sustainability has shown for years that a tiny percentage of contributors do the majority of review work. So, in open source, this kind of review almost always falls on the same small group of people. You see the same names over and over again on merged pull requests.“

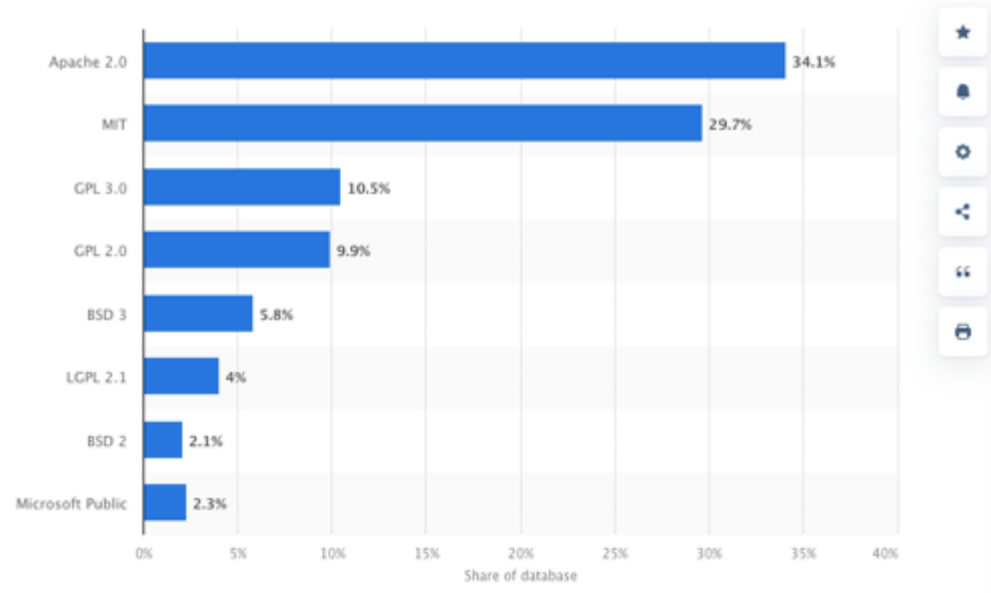


https://www.coderabbit.ai/blog/ai-is-burning-out-the-people-who-keep-open-source-alive

Software Licenses

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

Most popular open source licenses worldwide in 2021



Additional Information

Show source

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 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 derivates)
- 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