Open Source: Historical Context

17-313 Fall 2026

Foundations of Software Engineering

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

Administrivia

- P4B due tonight.
- P5 will be posted today.
- Next week's recitation: P5 kickstart presentation (5 min).

Learning Goals

- Describe how changes in hardware, software distribution, and commercial incentives shaped open source.
- Explain the motivations behind the free software movement and the emergence of the open source framing.
- Compare the major open-source licenses and the obligations each one imposes.
- Explain why companies use, release, and contribute to open-source software.
- Identify several ways to participate in an open-source community.

Outline

- **What Is Open Source?**
- Historical Context
- Open Source Licenses
- Why Companies Invest
- Participating in Open Source

We Are All Open Source Users



Phones, browsers, databases, the tools in your own projects: most of what you touch every day is open source.

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 and philosophical debate exists, and being informed is healthy.

How did software sharing become an organized movement and a major part of commercial software development?

What Is Open Source Software?

- The simplest definition: software released under an **open source license**.
- Such a license grants users the freedom to use, inspect, modify, and redistribute the software.
- The Open Source Definition (OSD) lists 10 traits a license must satisfy to count as "open source."
- The OSD is maintained by the Open Source Initiative (OSI), a nonprofit that judges licenses against it.
- Around 80 licenses are OSI-approved, but only a handful (MIT, Apache-2.0, GPL, BSD, MPL-2.0, …) are commonly used.

The Open Source Definition

The OSD lists attributes a software license must have. A few of the ten:

- **Free Redistribution**: no one is charged a fee to redistribute the software.
- **Source Code**: the program must include, or allow access to, its source code.
- **Derived Works**: modifications and derived works must be allowed, under the same license.
- **No Discrimination** against persons, groups, or fields of endeavor.

NOTES

Free redistribution means a license can't require a royalty to resell the software. Copyleft licenses require derived works to keep the same terms; permissive licenses leave that choice to whoever redistributes. Both still have to allow commercial use.

Outline

- What Is Open Source?
- **Historical Context**
- Open Source Licenses
- Why Companies Invest
- Participating in Open Source

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 is released as public domain.



NOTES

Software made expensive, shared computers more useful, so institutions had an incentive to exchange programs. Early compilers reflect a general push toward reusing programming effort rather than one single ownership model.

A Bit of History: 1960s

- IBM’s System/360 brings general-purpose computing.
- The rise of "minicomputers" (DEC PDP-8), alongside "IBM and the Seven Dwarfs."
- Software is usually bundled with hardware, and development is expensive.
- **1969: "software unbundling."** Software starts becoming a separately priced product, and an industry of its own.







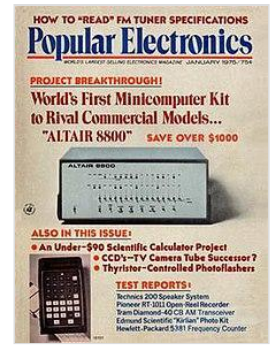
“IBM and the Seven Dwarfs”

NOTES

Bundling meant hardware prices quietly subsidized software. IBM's 1969 unbundling decision made many programs separately priced products, opening room for independent software companies to compete directly.

A Bit of History: 1970s

- Rise of the "microcomputers": PDP, Altair.
- Unix is first released by Bell Labs.
- Microsoft is founded.
- Bill Gates writes an "Open Letter to Hobbyists."

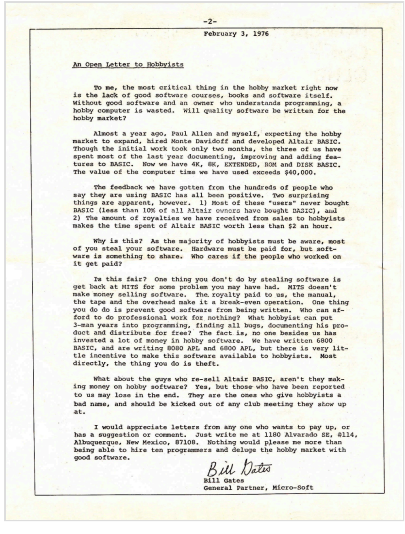




NOTES

Hobbyist communities and software businesses grew up side by side as computers reached more people. The Altair was a microcomputer, DEC's PDP line were minicomputers; Unix, Microsoft's founding, and Gates's letter all reflect this same expanding, increasingly commercial market.

Gates's "Open Letter to Hobbyists"



-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.

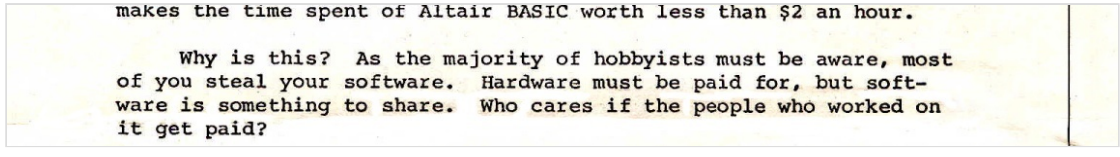
Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

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



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?

NOTES

Gates argued that widespread copying cut into the revenue needed to pay developers; hobbyists countered that sharing lowered costs and helped others use their machines. Both sides agreed development needs to be sustained, they just disagreed on how to fund it.

Tiny BASIC: "All Wrongs Reserved" @COPYLEFT

```
;*****************************************
;
;     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 hobbyist's wordplay on "copyright": software meant to be freely copied and shared, not restricted.

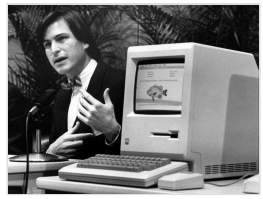
NOTES

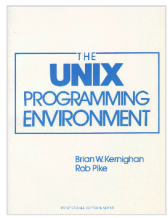
"All Wrongs Reserved" inverts "all rights reserved," capturing a culture that welcomed copying and adaptation. The GPL later turned that spirit into a legal mechanism: copyright permissions that come with conditions preserving the next recipient's freedoms.

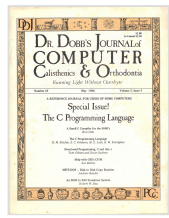
A Bit of History: 1980s

- Rise of personal computers; the software industry flourishes (Adobe, Autodesk, Lotus, …).
- Informal software sharing among "hobbyists" continues.
- *Apple Computer, Inc. v. Franklin Computer Corp.* tests copyright over software.
- *The Unix Programming Environment* is published.
- Stallman starts the Free Software Movement.









NOTES

As the software market grew, control over copying became commercially valuable. Stallman's 1983 GNU announcement proposed a complete, freely shareable Unix-compatible system, tying a freedom-based principle to the practical work of building an OS and recruiting contributors.

"Free as in Free Speech"



FREE SOFTWARE FOUNDATION

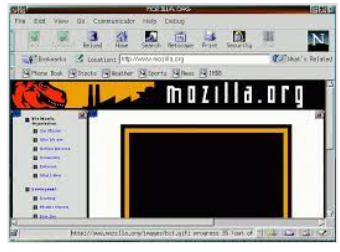
GNU/Linux

- Stallman starts the Free Software Movement: users should be free to run, study, modify, and share software.
- The Free Software Foundation (FSF) and the GNU project organize this into licenses, most notably the **GPL**.
- "Free" means freedom, not price, the "free as in free speech" distinction.

A Bit of History: 1990s

- **Linux** gains traction as a free, collaboratively developed operating system.
- Netscape starts the open source **Mozilla** project during the first browser war.
- The **Open Source Initiative** (OSI) forms to promote this model.







NOTES

Linux showed that distributed developers could collaborate on something substantial; Netscape's 1998 source release brought that model into open commercial competition for browsers. That same year, "open source" and the OSI reframed the same practices around practical, business-friendly benefits.

The Open Source Initiative



The OSI framed "open source" around practical benefits: collaboration, shared development, and adoption by businesses, connecting back to the OSD definition introduced earlier.

Old Perceptions

- Anarchy
- Demagoguery
- Ideology
- Altruism
- "Many eyes" (all bugs are shallow)



These were perceptions some observers held, not objective descriptions of open source.

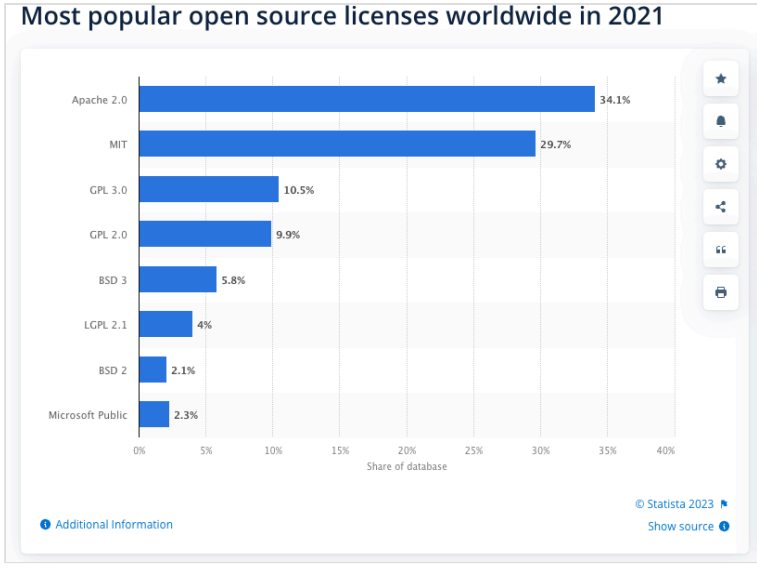
NOTES

These labels reflect real doubts about coordination and contributors' motives. "Many eyes" only works if people are actually reading the code, understanding it, and following through on fixes, so governance and maintenance effort matter as much as headcount.

Outline

- What Is Open Source?
- Historical Context
- **Open Source Licenses**
- Why Companies Invest
- Participating in Open Source

Open Source Licenses



Apache-2.0, MIT, and the GPL family are by far the most widely used.

The GNU GPL: The Copyleft License

Nobody should be restricted by the software they use. Four freedoms 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.
- The freedom to share the changes you make.

Code must be made available, and any modifications must be re-licensed under the same license (**copyleft**).

NOTES

The GPL uses copyright to guarantee recipients can use, modify, and redistribute the software; anyone distributing a modified version must pass those same rights along. Private, internal use creates no obligation to publish source: what matters is how the result is distributed.

GPL 2.0 and 3.0

GPLv2

- A court ruling cannot nullify the license: if a court decision and the license contradict on distribution requirements, 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 that block running modified software).

Warning: GPLv2 and GPLv3 have compatibility issues with each other.

NOTES

The term comes from TiVo: its DVRs ran GPL-licensed Linux and TiVo complied by publishing the source, but the hardware would refuse to boot any modified version, so the "freedom to run modified software" was technically legal but practically blocked. "Tivoization" means shipping source code while using hardware locks to block users from running their own modified version; GPLv3 closes that loophole and adds explicit patent terms. Mixing GPLv2-only code with GPLv3 code is a real compatibility problem unless the code allows "or any later version."

LGPL and BSD

LGPL

- The licensed software must be a library (a component used in other products).
- Similar to the GPL, but without the copyleft requirement on the software that links it.

BSD License

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

NOTES

LGPL is built for libraries: other licenses can link against it, but the library itself, including any modifications, stays copyleft. BSD is more permissive, just keeps notice requirements, and in its 3-clause form adds a no-endorsement clause.

MIT and Apache Licenses

MIT License

- Must retain copyright credit.
- Provided as is; authors are not liable.
- Unlike the GPL, does not require derivative works to 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 the project's trademark.
- Allows proprietary use, with an explicit grant of patent rights.

Two Extremes

- **No license** (default is "all rights reserved"): without an explicit license, none of the OSD freedoms apply.
- **Public domain dedication** (e.g., the Unlicense): the author gives up all rights; no attribution or conditions at all.

NOTES

No license at all means copyright defaults to "all rights reserved," leaving redistribution rights unclear. The Unlicense is the opposite extreme: public domain plus broad permission to copy, modify, and distribute, stated explicitly so other developers have a clear basis to reuse the code.

Dual License Business Model



- MySQL is released as GPL, which requires a company using the open source product to open source its own application.
- Or, companies can pay $2,000 to $10,000 annually to receive a copy of MySQL under a more business-friendly license.

NOTES

Dual licensing only works for whoever owns the copyright: Oracle acquired MySQL's rights through its purchase of Sun Microsystems, so it can offer the same code under GPL or under a paid commercial license. Users pick whichever track fits them: the GPL track suits projects willing to open-source in return, the paid track suits proprietary distributors.

Risk: Incompatible Licenses





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

NOTES

LibreOffice could pull in Apache-licensed OpenOffice code with its notices intact, but its own copyleft-licensed code can't flow back the other way into an Apache-only distribution. License compatibility can be a one-way street.

What Not to Do: Winamp

- Winamp, a popular media player, was open-sourced to revive community interest.
- The source release inadvertently included proprietary components from Microsoft, Intel, Dolby, and the SHOUTcast server software.
- The "Winamp Collaborative License" prohibited forks and distribution of modified versions, contradicting open-source principles.
- The release was also later found to include GPL-licensed components, creating a license violation.







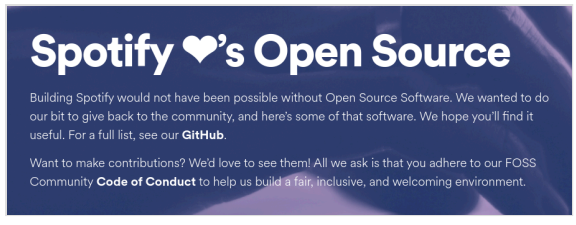
NOTES

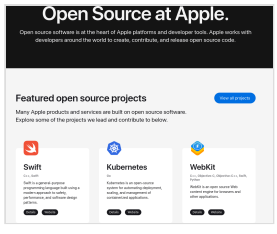
The Winamp release restricted forks and modified versions in ways that directly conflict with the Open Source Definition. Publishing an existing codebase also means checking which parts you actually have permission to redistribute, and under what terms, before you ship it.

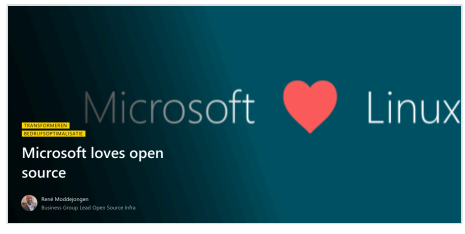
Outline

- What Is Open Source?
- Historical Context
- Open Source Licenses
- **Why Companies Invest**
- Participating in Open Source

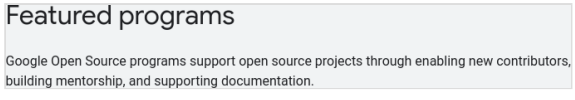
Is It Altruism?

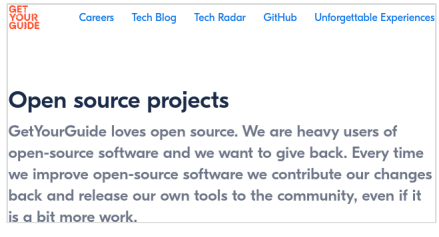












Spotify, Apple, Microsoft, NVIDIA, Google, and many others all publish pages about how much they "love" open source. Why?

Open Source as a Business

Some companies build their entire business around open-source software, rather than around a proprietary product. How?

- **Professional services** (Red Hat, Canonical): consulting and training. Risk: thin margins, renewal rates, service-only competition.
- **Open core** (GitLab, Databricks/Apache Spark, MongoDB): give away a base product, sell premium features. Risk: competition on the commercial add-ons.
- **Software as a service** (WordPress): host and operate it for customers. Risk: free-riding by cloud providers.

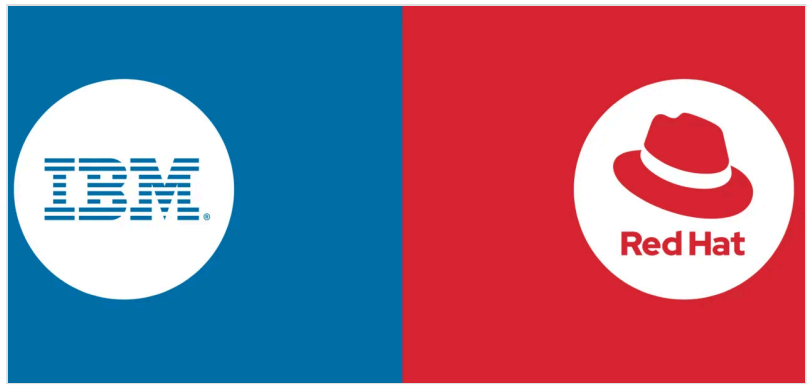








Red Hat: A Successful Example



Red Hat built a business entirely on open source professional services, eventually acquired by IBM: one of the clearest examples that the model can work at scale.

NOTES

Red Hat sells subscriptions, not the software itself: maintenance, security updates, support, and lifecycle commitments bundled around freely available code. Customers are really paying for a dependable, supported platform and access to expertise.

Build, Buy, or Something Smarter?

- Build versus buy is one of the most fundamental questions in engineering: the cost of building from scratch versus buying or licensing, plus the cost of maintenance over time.
- Open source adds a third option: **borrow and share**.

Borrow-and-Share

Add OSI-licensed projects, and you add a third option: **borrow** what communities have built, **adapt** it, **share** enough back to sustain it.

The numbers make the case:

- Kubernetes (Google) ≈ 4.6M lines of code. VS Code (Microsoft) ≈ 1.2M.
- Rule of thumb: 1M lines of code ≈ 100 engineers × 5 years ≈ **$75M** to build from scratch.
- In both projects, **about half of all contributions come from outside the founding company**.

NOTES

These figures are illustrative, not precise: lines of code is a rough cost proxy, and contribution share depends on when you measure it. The real argument is that shared infrastructure spreads development and maintenance cost across many organizations, though adopters still have to budget for integration and upkeep.

Is It Altruism?

- It isn’t about altruism: it’s about **economics**.
- It is economically an easier decision to borrow-and-share than to build or buy.
- You always get more than you give, economically.

NOTES

Maintaining a fix upstream is often cheaper than carrying a private patch forever, and contributing can also attract outside contributors, encourage adoption, and shape a project's direction. Ethical motivation and commercial incentive usually point the same way here, not opposite ways.

Activity: Build, Buy, or Borrow & Share

In pairs: discuss the scenario below. Decide whether you would build, buy, or borrow & share (open source). There is no single correct answer; the goal is to justify your choice.

Scenario: Internal Data Visualization Dashboard

Your team needs rich dashboards for internal analytics and monitoring. Build, buy, or borrow & share?

30s setup · 90s pairs decide · 90s share out · 30s connect back

Outline

- What Is Open Source?
- Historical Context
- Open Source Licenses
- Why Companies Invest
- **Participating in Open Source**

Understand Governance Styles

What can you do now to get started? First, understand how the project makes decisions.

Consensus

Decisions are made by agreement among contributors. Apache, Rust, and many others.

Benevolent Dictator

One person (or a small group) has final say. Python, Linux.

Corporate

A company controls the roadmap and direction.

NOTES

Governance is really about who can approve changes, set priorities, and cut releases; real projects often blend these styles. Python is the classic benevolent-dictator example but now runs on an elected five-person steering council, a reminder that a project's governance can change over time.

Contribute Early

What can you do now to get started?

- When you need to customize an open-source tool, try to contribute your changes back to the original project (the "**upstream**").
- This reduces your own maintenance burden: you don’t have to keep re-patching every time the project updates.
- It also builds reputation and influence inside communities that matter to your work.

Version Control Discipline

Clean history makes a contribution easier to review and merge.

- Small, focused commits with clear messages, not one giant changeset.
- A feature branch per change, rebased or merged cleanly before opening a pull request.
- Follow the project's own contribution guidelines: commit format, branch naming, and required CI checks.







Personal Economics

- Having your name on key contribution streams in an open-source community is some of the best resume content you can have: it shows you can get work done, work in a collaborative engineering setting, and understand a real technology base.
- The connections you make in a community expand your network, and it’s often the **weak ties** that make the difference in the job market.
- Reading known-good software projects is also a good way to improve your own skills.

Granovetter, "The Strength of Weak Ties" (1973).

NOTES

Weak ties, Granovetter's term for acquaintances rather than close collaborators, often carry information your close network doesn't have. Open-source contributions are a visible way to build exactly those ties, while also giving others concrete evidence of how you communicate and solve problems.

Contributions: Many Ways to Help

- **Code**: fixes and new functionality increase the project's value and broaden its use.
- **Bug reports** reveal a new test path or use case.
- **Documentation**: answering questions, tutorials, presentations, FAQs, broaden the user base.
- **Translations** of messages and documentation can make a big difference.
- Anything someone brings to the project counts.

Even for your own class projects: a clear bug report or a useful documentation improvement can be a meaningful contribution.

The Open Source Community: A Living Ecosystem



- Hardware and software business models shaped early sharing practices.
- The free software movement and open source institutions organized principles and practices around sharing.
- Companies and individuals participate for overlapping ethical, technical, professional, and economic reasons.
- Projects depend on the work of users, contributors, maintainers, creators, and sponsors.

Takeaways: What You Need to Know

- 1969’s "software unbundling" made software its own priced industry, out of a prior default of sharing.
- The free software movement and the OSI gave "open source" its meaning: freedoms and copyleft, later reframed for business.
- Licenses differ mainly on one axis, permissive versus copyleft, which sets your obligations.
- Companies build businesses around open source mostly for economic reasons: borrowing and sharing beats building or buying alone.
- Anyone can participate: code, bug reports, documentation, and translations all count as real contributions.