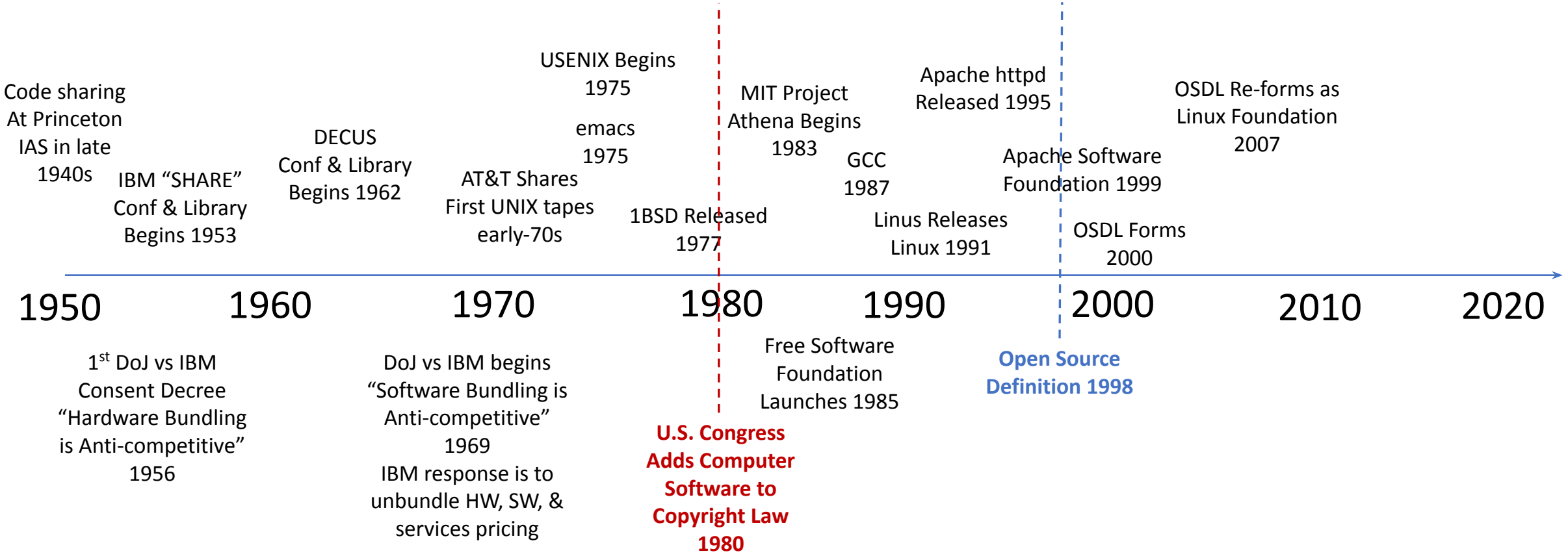
06 – **The Engineering Economics of Open Source Software**

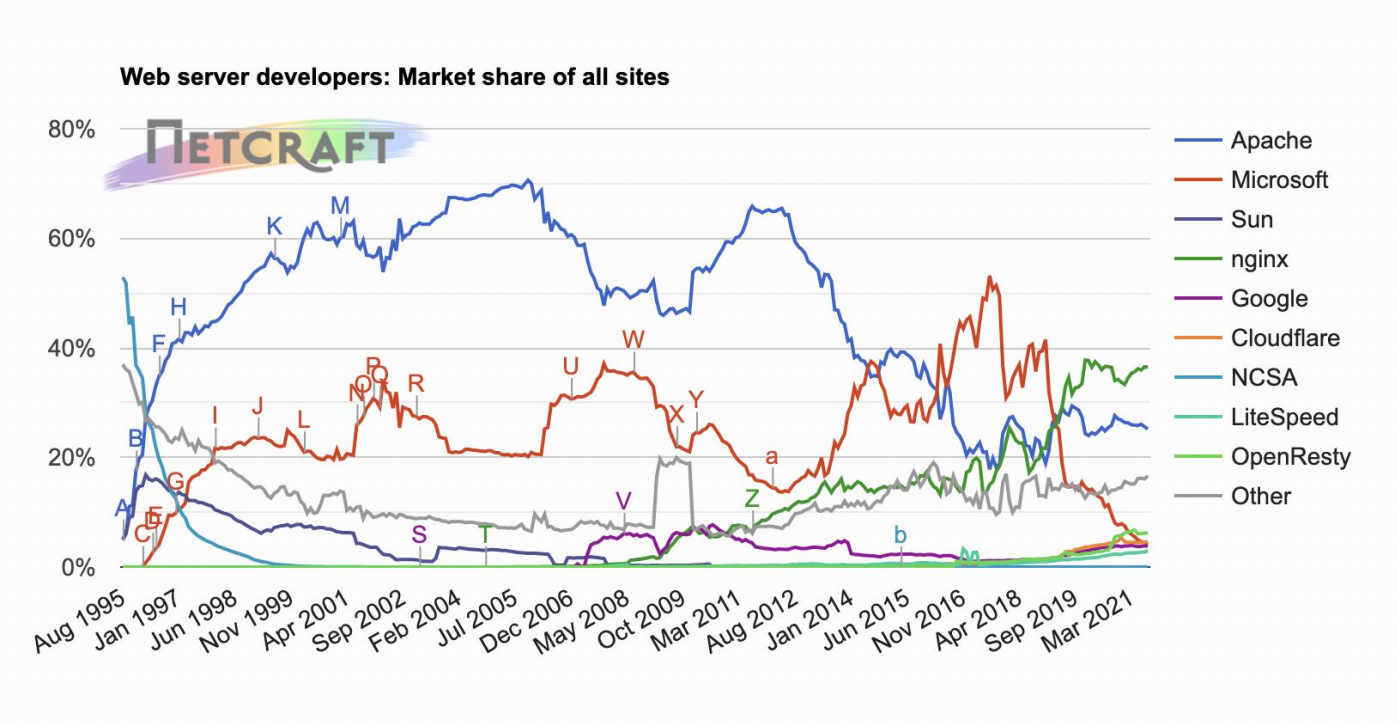
We've collaborated on software since we've created software (Because creating good software is hard work)



What is “Open Source Software”?

Let’s Build Apache httpd

- This is the classic webserver
- By 2000 roughly 60% - 70% of the world’s web traffic ran this way (now ~25%)
- Following the Lab 1 instructions from Intersession Course (JHU-EN.601.210)



https://news.netcraft.com/archives/category/web-server-survey/
https://github.com/jhu-ospo-courses/JHU-EN.601.210/tree/main/labs/1

The Demo …

[https://github.com/jhu-ospo-courses/JHU-EN.601.210/tree/main/labs/1]
[https://httpd.apache.org/docs/current/install.html]

Installing from source

Download	Download the latest release from http://httpd.apache.org/download.cgi
Extract	`$ gzip -d httpd-NN.tar.gz`
	`$ tar xvf httpd-NN.tar`
	`$ cd httpd-NN`
Configure	`$ ./configure --prefix=PREFIX`
Compile	`$ make`
Install	`$ make install`
Customize	`$ vi PREFIX/conf/httpd.conf`
Test	`$ PREFIX/bin/apachectl -k start`

NN must be replaced with the current version number, and *PREFIX* must be replaced with the filesystem path under which the server should be installed. If *PREFIX* is not specified, it defaults to `/usr/local/apache2`.

Each section of the compilation and installation process is described in more detail below, beginning with the requirements for compiling and installing Apache httpd.

Don't see your favorite platform mentioned here? Come help us improve this doc.

Requirements

The following requirements exist for building Apache httpd:

APR and APR-Util

Make sure you have APR and APR-Util already installed on your system. If you don't, or prefer to not use the system-provided versions, download the latest versions of both APR and APR-Util from Apache APR, unpack them into `/httpd_source_tree_root/srclib/apr` and `/httpd_source_tree_root/srclib/apr-util` (be sure the directory names do not have version numbers; for example, the APR distribution must be under /httpd_source_tree_root/srclib/apr/) and use `./configure`'s `--with-included-apr` option. On some platforms, you may have to install the corresponding `-dev` packages to allow httpd to build against your installed copy of APR and APR-Util.

Perl-Compatible Regular Expressions Library (PCRE)

This library is required but not longer bundled with httpd. Download the source code from http://www.pcre.org, or install a Port or Package. If your build system can't find the pcre-config script installed by the PCRE build, point to it using the `--with-pcre` parameter. On some platforms, you may have to install the corresponding `-dev` package to allow httpd to build against your installed copy of PCRE.

So what …

COCOMO RESULTS for Apache								
MODE	**"A" variable**	**"B" variable**	**"C" variable**	**"D" variable**	**KLOC**	**EFFORT, (in person-months)**	**DURATION, (in months)**	**STAFFING, (recommended)**
semi-detached	3	1.12	2.5	0.35	280.000	1651.727	33.436	49.399

Explanation: The coefficients are set according to the project mode selected on the previous page, (as per Boehm). Note: the decimal separator is a period.

The final estimates are determined in the following manner:

effort = a*KLOC^{b}, in person-months, with KLOC = lines of code, (in thousands), and:

staffing = effort/duration

where a has been adjusted by the factors:

280K LoC (refactored)
1650 months == 137.5 years
At $100K/year * 1.5 (gross up) == $150K
$20.6M of software value

https://strs.grc.nasa.gov/repository/forms/cocomo-calculation/
https://www.indeed.com/career/software-engineer/salaries

But … wait … it's BETTER than that …

- Apache httpd has been refactored – this build was just the base
- Linux (~50M LoC)
- Docker
- gcc
- configure (autoconf/automake), make
- Perl
- All of this is open source licensed and built in collaborative communities

Open Source Software Defined

- The simplest definition is that it is software that is published using 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 since 1998
- There are ~120 licenses, but only a handful are really 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

The Simplest OSI-Approved License (MIT)

Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

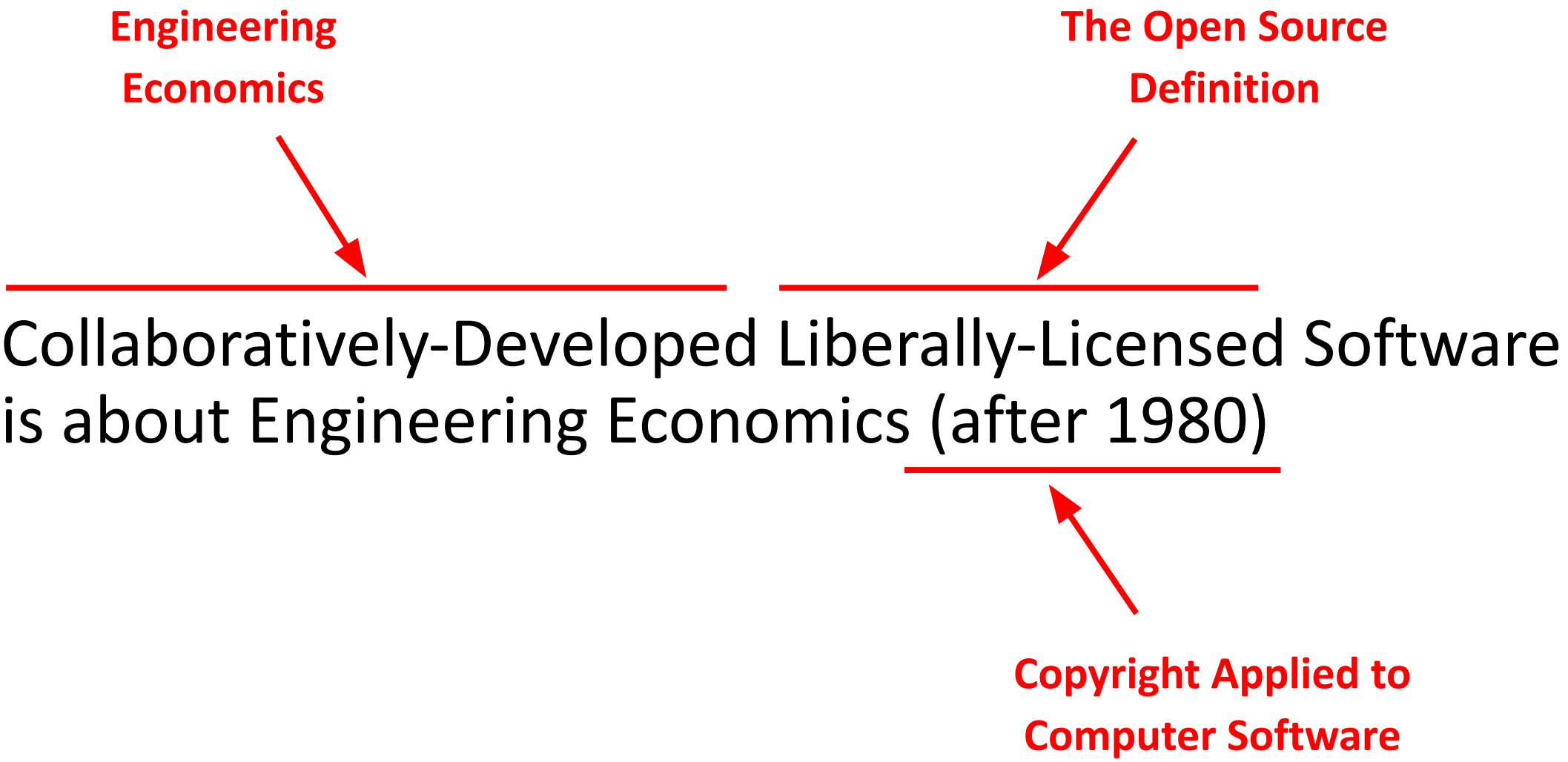
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The More Complex Answer
(To the Question, “What is Open Source Software?”)

- It is collaboratively developed in a community of like-minded people
- An innovator shares their work outward – **with no expectation of return** – BUT a willingness to accept contributions
- There is an “Architecture of Participation” for how others contribute

Eric von Hippel, “Open source projects as horizontal innovation networks - by and for users ”,
https://www.oecd.org/education/innovation-education/32125887.pdf
Tim O’Reilly coined the expression “Architecture of Participation”
https://www.oreilly.com/pub/a/tim/articles/paradigmshift_0504.html


Engineering Economics
The Open Source Definition
Collaboratively-Developed Liberally-Licensed Software is about Engineering Economics (after 1980)
Copyright Applied to Computer Software

Every Engineering/Product Manager understands the “**Build** versus **Buy**” decision

To Build or to Buy, that is the question …

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

Lines-of-Code & Boehm COCOMO

- Lines-of-code (LoC) is a horrible measure
- LoC is also an interesting relative measure
- Boehm's Constructive Cost Model (COCOMO) turned LoC and the team and aspects of the project into a (rough) value equation

The Interix Compiler Example
(An Open Source Consumption Example)

- Interix was the UNIX front-end on Windows NT (1995)
- ~300 software packages covered by ~30 licenses (pre-Open Source Definition)
- Running on Windows NT across Intel x86, DEC Alpha, MIPS CPUs.
- Needed to replace the Microsoft compiler base …

https://www.usenix.org/legacy/publications/library/proceedings/usenix-nt97/full_papers/walli/walli.pdf
https://medium.com/@stephenrwalli/running-linux-apps-on-windows-and-other-stupid-human-tricks-part-i-acbf5a474532

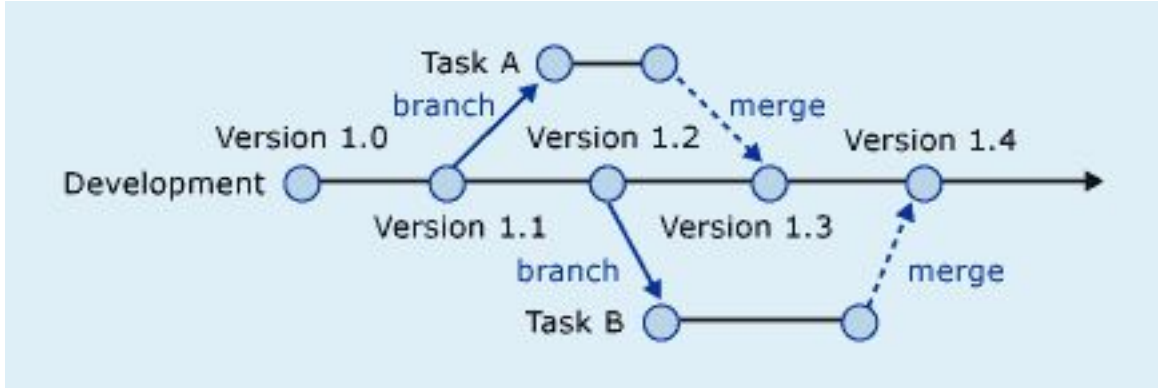
What’s the Build vs Buy for a Compiler?

- Venture funded startup with limited capital
- Build would involve turning the team to building a compiler on three computer architectures
- Buy would involve licensing+support from Intel and DEC costing hundreds of thousands (USD) and doesn’t solve for MIPS
- Borrow-and-share and the GNU Compiler Suite (gcc) – The Interix team hires a compiler expert and joins the gcc community

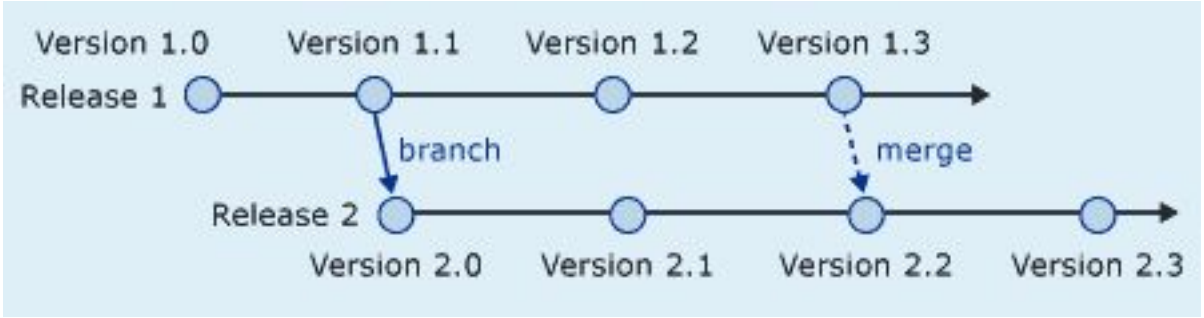
gcc costing (in 1997)

- Hire an experienced compiler developer ~$100K
- gcc runs on all architectures
- ~750K LoC == ~$10M-$20M (COCOMO) depending on cost/head
- Robust, hardened compiler including F77 and C++ front-ends (also Ada for those that cared in the late ‘90s)
- This is two orders of magnitude of value capture
- But … now we’re on a fork

A Digression on Branches and Forks

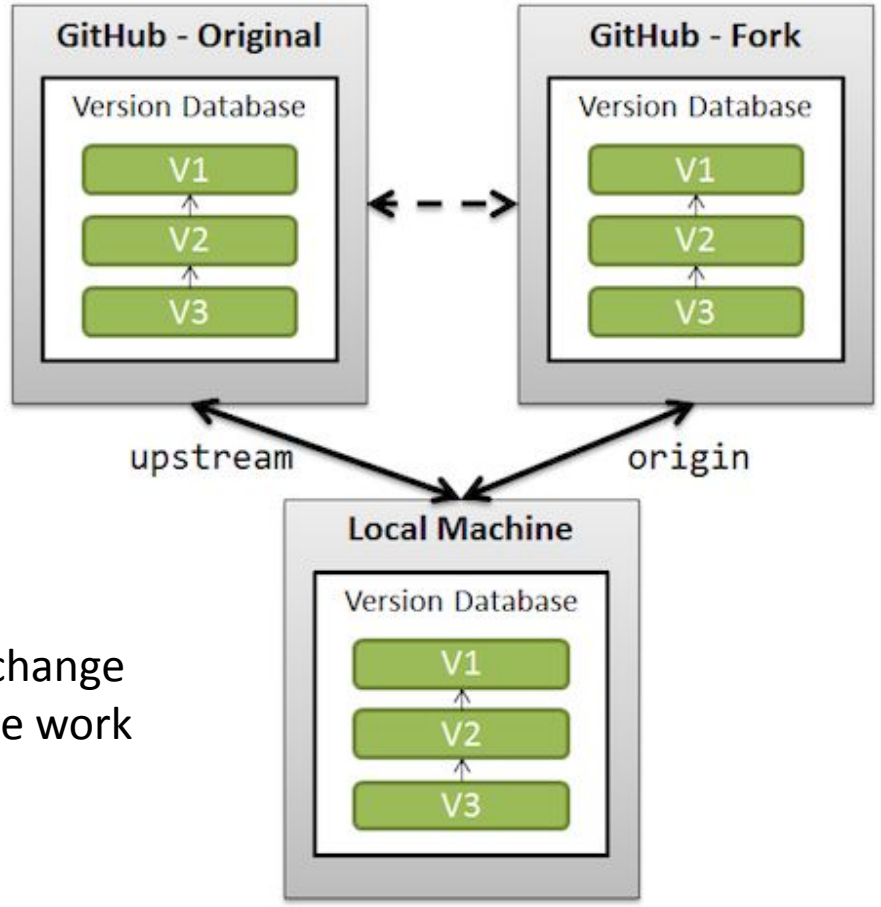


Branch per task strategy



Branch per release

Forks are more private/separate working copies



Branching is how to think about change
Forking is how you accomplish the work

gcc fork integration

- Every new major gcc release will cost 6 months of integration again!
- If we get our changes merged upstream it will be ~1 month of integration
- This is bringing integration costs into the $10K space
- BUT gcc community is really 5 open source communities – **some hostile!**
- The Interix engineering team hired gcc specialists (gcc maintainers) for $40K to merge to head rev
- The gcc project was evolving as well – by the time the Interix fork was added the gcc code base had grown to 1.2M LoC

Engineering Economics

- This wasn’t altruism – it was contributing back to the same hardening from which we benefited while managing engineering costs
- Economics works in the small – For a few lines of changes, a developer gets an entire webserver
- Economics works in a research community long-term as well
- The Booking Agents using Red Hat Linux while paying for JBoss

Side Notes

- 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

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

Let’s Summarize Consumption Economics

- Build vs Buy vs Borrow + Share as the fundamental use economics
- You ALWAYS get more than you give
- When consuming you can easily be gaining 1-3 **orders of magnitude** of software value
- The economics works from the smallest scale (personal) all the way up to the largest scale (complex projects like compilers and operating systems)

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

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

- Kubernetes = 4.6MLoC = ~$630M
- VS Code = 1.2 MLoC = ~$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)

Let's Evaluate An Open Source Project

Choose one of the following projects.

- Perl, a foundational scripting language for systems administration and building (early) dynamic web sites.
- Python, an important programming language in the scientific, machine learning, and research community (and other places).
- Node.js, a programming environment for server-side javascript programming.

Your Investigation

The search engine is your friend. Trying simple explicit searches is best.

For example:

```
<project name> open source
build <project name> from source
<project name> license
<project name> meetup
```

The User On-Ramp

1. Project Website: Note project website URL.
2. Project License: Note project license URL. How many licenses are there?
3. Introductory Documents: Were there getting started docs or tutorials?
4. FAQ: Was there an FAQ?
5. User Installation: Does it look like you can install the project executables without building the project from source?
6. What platforms are supported?
7. Platform Installer Support?
8. Bugtracking/Issues Tracking?
9. How-to: Tutorials?

The Developer On-Ramp

10. **Source Code easy to find?**
11. **Are there good instructions for how to build the project?**
12. **Can you test to a Known State?**
13. Design Documentation?
14. Project Communications: IRC/Slack channel, any email distribution lists, or forums?
15. Is there a mission statement for the project?
16. Is there a code-of-conduct?

The Contributor On-Ramp

17. Could you find getting involved instructions or?
18. Could you find contribution guidelines?
19. Are there conferences or meet-ups?
20. Source Code Base: Use cloc to determine how big the source code base is.

In purely subjective terms, how did you find the whole experience?

All of this assumes that we are talking about healthy OSI-licensed projects …