Free and Open-source Software

Martin Kellogg

Free and Open-source Software

Today’s agenda:

- Finish static analysis slides
- Reading Quiz
- History + the “free software” philosophy
- Open-source: licenses and business models
- Mid-semester survey: how am I doing?

Free and Open-source Software

Today's agenda:

- Finish static analysis slides
- Reading Quiz
- History + the "free software
- Open-source: licenses and b
- Mid-semester survey: how an

Announcements

- reminder: optional reading #1 due soon (Saturday night)
- we plan return all graded revised project plans by Friday evening

Free and Open-source Software

Today's agenda:

- **Finish static analysis slides**
- Reading Quiz
- History + the "free software" philosophy
- Open-source: licenses and business models
- Mid-semester survey: how am I doing?

Limitations of static analysis

- static analysis **abstracts away** information to remain decidable

Limitations of static analysis

- static analysis **abstracts away** information to remain decidable
  - **potential problem**: what if the information that was abstracted away is important?

Limitations of static analysis

- static analysis **abstracts away** information to remain decidable
  - **potential problem**: what if the information that was abstracted away is important?
    - can we come up with a program for which one of our example static analyses “gets the wrong answer”?

Limitations of static analysis

- static analysis **abstracts away** information to remain decidable
    - **potential problem**: what if the information that was abstracted away is important?
        - can we come up with a program for which one of our example static analyses “gets the wrong answer”?
    - can we ever have a “**perfect**” abstraction?

Limitations of static analysis

- static analysis **abstracts away** information to remain decidable
  - **potential problem**: what if the information that was abstracted away is important?
    - can we come up with a program for which one of our example static analyses "gets the wrong answer"?
  - can we ever have a "**perfect**" abstraction?
    - of course not (Rice's theorem again)

Limitations of static analysis

- static analysis **abstracts away** information to remain decidable
  - **potential problem**: what if the information that was abstracted away is important?
    - can we come up with a program for which one of our example static analyses “gets the wrong answer”?
  - can we ever have a “**perfect**” abstraction?
    - of course not (Rice’s theorem again)
    - but, in practice, we can get very close

Limitations of static analysis

- static analysis is **best** when the rules it enforces are:

Limitations of static analysis

- static analysis is **best** when the rules it enforces are:
  - simple to express to the computer
  - hard for a human to apply

Limitations of static analysis

- static analysis is **best** when the rules it enforces are:
    - simple to express to the computer
    - hard for a human to apply
- **implication**: if you find yourself struggling to follow a well-defined (but complicated for a human) rule set while writing code, it might be time to reach for a static analysis

Limitations of static analysis

- static analysis is **best** when the rules it enforces are:
  - simple to express to the computer
  - hard for a human to apply
- **implication**: if you find yourself struggling to follow a well-defined (but complicated for a human) rule set while writing code, it might be time to reach for a static analysis
  - this sort of situation comes up often:

Limitations of static analysis

- static analysis is **best** when the rules it enforces are:
  - simple to express to the computer
  - hard for a human to apply
- **implication**: if you find yourself struggling to follow a well-defined (but complicated for a human) rule set while writing code, it might be time to reach for a static analysis
  - this sort of situation comes up often:
    - x86/64 calling convention

Limitations of static analysis

- static analysis is **best** when the rules it enforces are:
  - simple to express to the computer
  - hard for a human to apply
- **implication**: if you find yourself struggling to follow a well-defined (but complicated for a human) rule set while writing code, it might be time to reach for a static analysis
  - this sort of situation comes up often:
    - x86/64 calling convention
    - complex API protocols (“call A then B then C then …”)

Limitations of static analysis

- static analysis is **best** when the rules it enforces are:
  - simple to express to the computer
  - hard for a human to apply
- **implication**: if you find yourself struggling to follow a well-defined (but complicated for a human) rule set while writing code, it might be time to reach for a static analysis
  - this sort of situation comes up often:
    - x86/64 calling convention
    - complex API protocols (“call A then B then C then …”)
    - security rules, etc.

Static analysis in practice

You’re likely to encounter:

Static analysis in practice

You’re likely to encounter:

- static **type systems** (sound)

Static analysis in practice

You're likely to encounter:

- static **type systems** (sound)
- **linters** or other style checkers (**syntactic** = not dataflow)

Static analysis in practice

You're likely to encounter:

- static **type systems** (sound)
- **linters** or other style checkers (**syntactic** = not dataflow)
- "***heuristic***" bug-finding tools backed by dataflow analyses

Static analysis in practice

heuristic is a fancy word for "best effort"

You're likely to encounter:

- static **type systems** (sound)
- **linters** or other style checkers (**syntactic** = not dataflow)
- "***heuristic***" bug-finding tools backed by dataflow analyses

Static analysis in practice

You’re likely to encounter:

- static **type systems** (sound)
- **linters** or other style checkers (**syntactic** = not dataflow)
- “***heuristic***” bug-finding tools backed by dataflow analyses
    - built into modern IDEs

Static analysis in practice

You're likely to encounter:

- static **type systems** (sound)
- **linters** or other style checkers (**syntactic** = not dataflow)
- "***heuristic***" bug-finding tools backed by dataflow analyses
  - built into modern IDEs
  - aim for low false positive rates

Static analysis in practice

You're likely to encounter:

- static **type systems** (sound)
- **linters** or other style checkers (**syntactic** = not dataflow)
- "***heuristic***" bug-finding tools backed by dataflow analyses
  - built into modern IDEs
  - aim for low false positive rates
  - widely used in industry:
    - ErrorProne at Google, Infer at Meta, SpotBugs at many places (including Amazon), Coverity, Fortify, etc.

Static analysis in practice

Less common, but useful to know about:

Static analysis in practice

Less common, but useful to know about:

- ***pluggable*** type systems

Static analysis in practice

Less common, but useful to know about:

- ***pluggable*** type systems
  - these are extensions to a type system that lets it prove more properties, e.g., adding nullness-checking to Java

Static analysis in practice

Less common, but useful to know about:

- ***pluggable*** type systems
  - these are extensions to a type system that lets it prove more properties, e.g., adding nullness-checking to Java
  - most common sound analysis (used by Google, Uber, others)

What is a pluggable type?

```
@Positive int x
```

What is a pluggable type?

$$\texttt{@Positive}\ \underbrace{\texttt{int}}_{\text{Basetype}}\ \texttt{x}$$

What is a pluggable type?

$$\underbrace{\texttt{@Positive}}_{\text{Type qualifier}}\ \underbrace{\texttt{int}}_{\text{Basetype}}\ \texttt{x}$$

What is a pluggable type?

$$\underbrace{\texttt{@Negative}}_{\text{Type qualifier}}\ \underbrace{\texttt{int}}_{\text{Basetype}}\ \texttt{x}$$

What is a pluggable type?

$$\underbrace{\texttt{@NonConstant}}_{\text{Type qualifier}}\ \underbrace{\texttt{int}}_{\text{Basetype}}\ \texttt{x}$$

What is a pluggable type?

$$\underbrace{\texttt{@Positive}}_{\text{Type qualifier}}\ \underbrace{\texttt{int}}_{\text{Basetype}}\ \texttt{x}$$

What is a pluggable type?

$$\underbrace{\underbrace{\texttt{@Positive}}_{\text{Type qualifier}}\ \underbrace{\texttt{int}}_{\text{Basetype}}}_{\text{Qualified type}}\ \texttt{x}$$

Pluggable type systems: key ideas

Pluggable type systems: key ideas

- developers already use static type systems, so they’re familiar with the general idea of types => **relatively easy to use** (compared to other sound static analyses)

Pluggable type systems: key ideas

- developers already use static type systems, so they’re familiar with the general idea of types => **relatively easy to use** (compared to other sound static analyses)
- type qualifiers **encode** property of interest
  - effectively a “second” type system

Pluggable type systems: key ideas

- developers already use static type systems, so they're familiar with the general idea of types => **relatively easy to use** (compared to other sound static analyses)
- type qualifiers **encode** property of interest
  - effectively a "second" type system
- qualified types are a **Cartesian product** of a type from the pluggable type system and a type from the base type system

Pluggable type systems: key ideas

- developers already use static type systems, so they're familiar with the general idea of types => **relatively easy to use** (compared to other sound static analyses)
- type qualifiers **encode** property of interest
  - effectively a "second" type system
- qualified types are a **Cartesian product** of a type from the pluggable type system and a type from the base type system
- typechecking is naturally **modular** = fast
  - but this comes at a cost: programmers need to write types

Pluggable type systems: key ideas

- developers already use static type ith
  the general idea of types => **relativ**
  other sound static analyses)
- type qualifiers **encode** property of
  - effectively a “second” type syst
- qualified types are a **Cartesian produ**
  pluggable type system and a type from the base type system
- typechecking is naturally **modular** = fast
  - but this comes at a cost: programmers need to write types

designing better (more expressive, more usable, etc.) pluggable type systems is an area of active research (mine!)

Static analysis in practice

Less common, but useful to know about:

- *pluggable* type systems
  - these are extensions to a type system that lets it prove more properties, e.g., adding nullness-checking to Java
  - most common sound analysis (used by Google, Uber, others)
- *formal verification* (subject of 3/7 reading)

Static analysis in practice

Less common, but useful to know about:

- *pluggable* type systems
  - these are extensions to a type system that lets it prove more properties, e.g., adding nullness-checking to Java
  - most common sound analysis (used by Google, Uber, others)
- *formal verification* (subject of 3/7 reading)
  - you write a specification

Static analysis in practice

Less common, but useful to know about:

- *pluggable* type systems
  - these are extensions to a type system that lets it prove more properties, e.g., adding nullness-checking to Java
  - most common sound analysis (used by Google, Uber, others)
- *formal verification* (subject of 3/7 reading)
  - you write a specification
  - tool verifies that code matches that specification

Static analysis in practice

Less common, but useful to know about:

- *pluggable* type systems
  - these are extensions to a type system that lets it prove more properties, e.g., adding nullness-checking to Java
  - most common sound analysis (used by Google, Uber, others)
- *formal verification* (subject of 3/7 reading)
  - you write a specification
  - tool verifies that code matches that specification
  - very high effort, but enables sound reasoning about complex properties (= worth it for very high value systems)

Static analysis in practice: soundiness

- all “**sound**” static analyses have a ***trusted computing base*** (***TCB***)

Static analysis in practice: soundiness

- all “**sound**” static analyses have a ***trusted computing base*** (*TCB*)
    - the TCB is the code whose correctness must be assumed for the analysis to actually be sound

Static analysis in practice: soundiness

- all “**sound**” static analyses have a ***trusted computing base*** (*TCB*)
  - the TCB is the code whose correctness must be assumed for the analysis to actually be sound
- **TCB size** is an important differentiator between “sound” analyses

Static analysis in practice: soundiness

- all “**sound**” static analyses have a ***trusted computing base*** (*TCB*)
  - the TCB is the code whose correctness must be assumed for the analysis to actually be sound
- **TCB size** is an important differentiator between “sound” analyses
  - e.g., TCB for many of my pluggable type systems includes the entire Java compiler (limits soundness a lot!)

Static analysis in practice: soundiness

- all “**sound**” static analyses have a ***trusted computing base*** (*TCB*)
    - the TCB is the code whose correctness must be assumed for the analysis to actually be sound
- **TCB size** is an important differentiator between “sound” analyses
    - e.g., TCB for many of my pluggable type systems includes the entire Java compiler (limits soundness a lot!)
    - TCB for some formal verifiers is **very small** (< 1000 LoC)
        - but these tools (e.g., Coq) are **much harder to use**

Static analysis in practice: soundiness

- all "**sound**" static analyses have a ***trusted computing base*** (*TCB*)
    - the TCB is the code whose correctness must be assumed for the analysis to actually be sound
- **TCB size** is an important differentiator between "sound" analyses
    - e.g., TCB for many of my pluggable type systems includes the entire Java compiler (limits soundness a lot!)
    - TCB for some formal verifiers is **very small** (< 1000 LoC)
        - but these tools (e.g., Coq) are **much harder to use**
- soundness theorems also usually make some **assumptions** about the code being analyzed (e.g., no calls to native code, no reflection)

Static analysis: summary

- static analysis is very good at enforcing **simple rules**
  - **much** better than humans at this
- all interesting semantic properties of programs are **undecidable**, so all static analyses must **approximate**
  - goal in analysis design is to **abstract away unimportant details**, but keep important details
  - **dataflow analysis** is one technique for static analysis
  - trade-offs between false positives, false negatives, analysis time
- soundness & completeness are **possible, but rare**
  - all soundness guarantees come with caveats about the TCB

Free and Open-source Software

Today’s agenda:

- Finish static analysis slides
- **Reading Quiz**
- History + the “free software” philosophy
- Open-source: licenses and business models
- Mid-semester survey: how am I doing?

Reading quiz: free and open-source software

Q1: The author claims that the term “open source software” means:

A. software you can get for zero price
B. software which gives the user certain freedoms
C. software whose source code you can look at
D. none of the above

Q2: The author claims that the term “free software” means:

- same options (**A**, **B**, **C**, **D**) as Q1

Reading quiz: free and open-source software

Q1: The author claims that the term “open source software” means:

A. software you can get for zero price
B. software which gives the user certain freedoms
C. software whose source code you can look at
D. none of the above

Q2: The author claims that the term “free software” means:

- same options (**A**, **B**, **C**, **D**) as Q1

Reading quiz

Q1: The author cla

A. software you c
B. software which
C. software whos
D. none of the abo

The official definition of open source software (... too long to include here) was derived indirectly from our criteria for free software. It is not the same; … However, the **obvious meaning** for … "open source software" is "You can look at the source code." Indeed, most people seem to **misunderstand** "open source software" that way.

Q2: The author claims that the term "free software" means:

- same options (**A**, **B**, **C**, **D**) as Q1

Reading quiz: free and open-source software

Q1: The author claims that the term “open source software” means:

A. software you can get for zero price
B. software which gives the user certain freedoms
C. software whose source code you can look at
D. none of the above

Q2: The author claims that the term “free software” means:

- same options (**A**, B, **C**, **D**) as Q1

Free and Open-source Software

Today's agenda:

- Finish static analysis slides
- Reading Quiz
- **History + the "free software" philosophy**
- Open-source: licenses and business models
- Mid-semester survey: how am I doing?

Why does this matter?

Why does this matter?

- Part of being a **software engineer** (vs just a programmer) is understanding the context of your work

Why does this matter?

- Part of being a **software engineer** (vs just a programmer) is understanding the context of your work
- “Free” vs “open-source” vs “closed-source”/”proprietary” is an important **philosophical debate** within the larger software engineering community

Why does this matter?

- Part of being a **software engineer** (vs just a programmer) is understanding the context of your work
- “Free” vs “open-source” vs “closed-source”/”proprietary” is an important **philosophical debate** within the larger software engineering community
- This debate has **consequences** for both how you build and how you use software that, as a software engineer, you should understand

Why does this matter?

- Part of being a **software engineer** (vs just a programmer) is understanding the context of your work
- "Free" vs "open-source" vs "closed-source"/"proprietary" is an important **philosophical debate** within the larger software engineering community
- This debate has **consequences** for both how you build and how you use software that, as a software engineer, you should understand
  - plus, it's the sort of thing that other, more senior engineers will expect you to have an **informed opinion** about

What is “open-source”?

What is “open-source”?

Definition: ***open source*** refers to any source code that is made freely available for possible modification and redistribution [Wikipedia]

What is “open-source”?

Definition: ***open source*** refers to any source code that is made freely available for possible modification and redistribution [Wikipedia]

- “open source” != “open source software” (we’ll talk about why later)

What is “open-source”?

Definition: ***open source*** refers to any source code that is made freely available for possible modification and redistribution [Wikipedia]

- “open source” != “open source software” (we’ll talk about why later)
- I’ll abbreviate “open source software” as **OSS**

The Case against Open Source

WHEN YOU PROGRAM OPEN SOURCE,
YOU'RE PROGRAMMING
COMMUNISM
A REMINDER
from
YOUR FRIENDS AT MICROSOFT

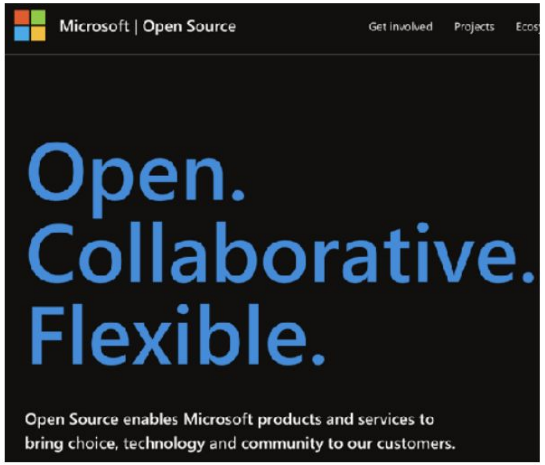
[Variation of popular meme, original source unknown]

The Case against Open Source

- “Open-Source Doomsday”: Once all software is free, we’ll stop making more software and have a market collapse
- Innovation will be stifled by the risk that software will be copied
- Making source code public means easier to attack
- “Anarchistic” licensing prevents companies from profiting from open source software

[Variation of popular meme, original source unknown]

The Case for Open Source


Microsoft | Open Source

Get involved Projects Ecos

Open. Collaborative. Flexible.

Open Source enables Microsoft products and services to bring choice, technology and community to our customers.

[Screenshot, 2022, opensource.microsoft.com]

The Case for Open Source

- Many eyes make all bugs shallow
- End-users can improve and customize software to their needs
- New features can be proposed and developed organically
- Greater productivity when more code is reused (easier with open source)
  - i.e., DRY on an industry-wide scale

Open. Collaborative. Flexible.

Open Source enables Microsoft products and services to bring choice, technology and community to our customers.

[Screenshot, 2022, opensource.microsoft.com]

History: open-source

History: open-source

- in the early days of computing, innovation **focused on hardware**

History: open-source

- in the early days of computing, innovation **focused on hardware**
    - no one was worried about keeping their code secret, since it usually would only run on their hardware anyway

History: open-source

- in the early days of computing, innovation **focused on hardware**
    - no one was worried about keeping their code secret, since it usually would only run on their hardware anyway
- what software development did occur happened mostly in **academic labs**, and AT&T's Bell Research Labs

History: open-source

- in the early days of computing, innovation **focused on hardware**
    - no one was worried about keeping their code secret, since it usually would only run on their hardware anyway
- what software development did occur happened mostly in **academic labs**, and AT&T’s Bell Research Labs
- Unix created at Bell Labs using the **new, portable** language “C” (~1970), licenses initially released with source code

History: open-source

- in the early days of computing, innovation **focused on hardware**
  - no one was worried about keeping their code secret, since it usually would only run on their hardware anyway
- what software development did occur happened mostly in **academic labs**, and AT&T's Bell Research Labs
- Unix created at Bell Labs using the **new, portable** language "C" (~1970), licenses initially released with source code
  - Unix quickly gained a lot of popularity for two reasons:

History: open-source

- in the early days of computing, innovation **focused on hardware**
    - no one was worried about keeping their code secret, since it usually would only run on their hardware anyway
- what software development did occur happened mostly in **academic labs**, and AT&T's Bell Research Labs
- Unix created at Bell Labs using the **new, portable** language "C" (~1970), licenses initially released with source code
    - Unix quickly gained a lot of popularity for two reasons:
        - portable between hardware (just need a C compiler)

History: open-source

- in the early days of computing, innovation **focused on hardware**
    - no one was worried about keeping their code secret, since it usually would only run on their hardware anyway
- what software development did occur happened mostly in **academic labs**, and AT&T's Bell Research Labs
- Unix created at Bell Labs using the **new, portable** language "C" (~1970), licenses initially released with source code
    - Unix quickly gained a lot of popularity for two reasons:
        - portable between hardware (just need a C compiler)
        - Bell Labs practically gave it away to universities

History: Unix

- 1978: UC Berkeley begins distributing their own derived version of Unix (BSD)

History: Unix

- 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

History: Unix

- 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
- 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)

The Free Software Philosophy

- UNIX distributed with source code, but with a **restrictive license**

The Free Software Philosophy

- UNIX distributed with source code, but with a **restrictive license**
- The Free Software Foundation promoted four “**freedoms**”:

The Free Software Philosophy

- UNIX distributed with source code, but with a **restrictive license**
- The Free Software Foundation promoted four "**freedoms**":

0. The freedom to run the program as you wish, for any purpose

The Free Software Philosophy

- UNIX distributed with source code, but with a **restrictive license**
- The Free Software Foundation promoted four "**freedoms**":

0. The freedom to run the program as you wish, for any purpose
1. The freedom to study how the program works, and change it so it does your computing as you wish

The Free Software Philosophy

- UNIX distributed with source code, but with a **restrictive license**
- The Free Software Foundation promoted four "**freedoms**":

  0. The freedom to run the program as you wish, for any purpose
  1. The freedom to study how the program works, and change it so it does your computing as you wish
  2. The freedom to redistributed copies (of the original) so you can help others

The Free Software Philosophy

- UNIX distributed with source code, but with a **restrictive license**
- The Free Software Foundation promoted four "**freedoms**":
    0. The freedom to run the program as you wish, for any purpose
    1. The freedom to study how the program works, and change it so it does your computing as you wish
    2. The freedom to redistributed copies (of the original) so you can help others
    3. The freedom to distribute copies of your modified version to others

The Free Software Philosophy

- UNIX distributed with source code, but with a **restrictive license**
- The Free Software Foundation promoted four "**freedoms**":

0. The freedom to run the program as you wish, for any purpose
1. The freedom to study how the program works, and change it so it does your computing as you wish
2. The freedom to redistributed copies (of the original) so you can help others
3. The freedom to distribute copies of your modified version to others

"Free as in speech, not as in beer"

The Free Software Philosophy

- the FSF claims: Free software should be licensed under the GNU Public License (GPL), considering questions like:

The Free Software Philosophy

- the FSF claims: Free software should be licensed under the GNU Public License (GPL), considering questions like:
  - Are you required to redistribute any modifications (under same license) - “***copyleft***”

The Free Software Philosophy

- the FSF claims: Free software should be licensed under the GNU Public License (GPL), considering questions like:
  - Are you required to redistribute any modifications (under same license) - "***copyleft***"
  - Can you redistribute executable binaries, or only source?

The Free Software Philosophy

- the FSF claims: Free software should be licensed under the GNU Public License (GPL), considering questions like:
  - Are you required to redistribute any modifications (under same license) - "***copyleft***"
  - Can you redistribute executable binaries, or only source?
  - Are you allowed to use the software in a restrictive hardware environment? ("***tivoization***")

The Free Software Philosophy

- the FSF claims: Free software should be licensed under the GNU Public License (GPL), considering questions like:
  - Are you required to redistribute any modifications (under same license) - "***copyleft***"
  - Can you redistribute executable binaries, or only source?
  - Are you allowed to use the software in a restrictive hardware environment? ("***tivoization***")

Difference between GPL v2 and GPL v3: is tivoization banned?

The Free Software Philosophy

- the FSF claims: Free software should be licensed under the GNU Public License (GPL), considering questions like:
  - Are you required to redistribute any modifications (under same license) - "***copyleft***"
  - Can you redistribute executable binaries, or only source?
  - Are you allowed to use the software in a restrictive hardware environment? ("***tivoization***")
- Popular alternative: "Do whatever you want with this software, but don't blame me if it doesn't work" ("***freeware***")

History: GNU/Linux (1991-Today)

- Stallman (FSF founder) set out to build an operating system in 1983, ended up building a **tremendous set of utilities** (“**GNU coreutils**”) that are needed by an OS (compiler, utilities, etc)

History: GNU/Linux

Remember: 1983 = Unix licensing changed because of AT&T breakup

- Stallman (FSF founder) set out to build an operating system in 1983, ended up building a **tremendous set of utilities** (“**GNU coreutils**”) that are needed by an OS (compiler, utilities, etc)

History: GNU/Linux (1991-Today)

- Stallman (FSF founder) set out to build an operating system in 1983, ended up building a **tremendous set of utilities** (“**GNU coreutils**”) that are needed by an OS (compiler, utilities, etc)
- **Linux** is an operating system built around and with the GNU utilities, licensed under GPL

History: GNU/Linux (1991-Today)

- Stallman (FSF founder) set out to build an operating system in 1983, ended up building a **tremendous set of utilities** (“**GNU coreutils**”) that are needed by an OS (compiler, utilities, etc)
- **Linux** is an operating system built around and with the GNU utilities, licensed under GPL
- Rise of the internet, demand for internet servers drives demand for cheap/free OS

History: GNU/Linux (1991-Today)

- Stallman (FSF founder) set out to build an operating system in 1983, ended up building a **tremendous set of utilities** (“**GNU coreutils**”) that are needed by an OS (compiler, utilities, etc)
- **Linux** is an operating system 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 began **adopting and supporting** Linux for enterprise customers: e.g., IBM committed over $1B; Red Hat and others

The Cathedral and the Bazaar (1997)

- Eric S Raymond’s influential 1997 essay compares two software development methodologies for OSS: “cathedral” or “bazaar”

The Cathedral and the Bazaar (1997)

- Eric S Raymond’s influential 1997 essay compares two software development methodologies for OSS: “cathedral” or “bazaar”
- “***cathedral***” model, where **releases** are available for anyone to see, but the development process is restricted to insiders

The Cathedral and the Bazaar (1997)

- Eric S Raymond's influential 1997 essay compares two software development methodologies for OSS: "cathedral" or "bazaar"
- "***cathedral***" model, where **releases** are available for anyone to see, but the development process is restricted to insiders
- However, most of the open source software ecosystem today follows the "***bazaar***" model:
  - Users treated as co-developers
  - Release software early for feedback
  - Modularize + reuse components
  - Democratic organization

The Cathedral and the Bazaar (1997)

- Eric S Raymond's influential 1997 essay compares two software development methodologies for OSS: "cathedral" or "bazaar"
- "***cathedral***" model, where **releases** are available for anyone to see, but the development process is restricted to insiders
- However, most of the open source software ecosystem today follows the "***bazaar***" model:
  - Users treated as co-developers
  - Release software early for feedback
  - Modularize + reuse components
  - Democratic organization

How did the bazaar model become dominant is OSS?

History: Netscape’s “Collaborating with the Net”

- **Netscape** was the dominant web browser in the early 90’s
    - Business model: free for home and education use, companies paid to use it

History: Netscape's "Collaborating with the Net"

- **Netscape** was the dominant web browser in the early 90's
  - Business model: free for home and education use, companies paid to use it
- Microsoft entered browser market with **Internet Explorer**, bundled with Windows in 1995, soon overtakes Netscape in usage (it's free, with Windows!)
  - also sued by US DoJ for antitrust bundling (!)

History: Netscape’s “Collaborating with the Net”

- Netscape was the dominant web browser in the early 90’s
  - Business model: free for home and education use, companies paid to use it
- Microsoft entered browser market with **Internet Explorer**, bundled with Windows in 1995, soon overtakes Netscape in usage (it’s free, with Windows!)
  - also sued by US DoJ for antitrust bundling (!)
- January 1998: Netscape becomes first (?) company to make **source code for proprietary product open** (Mozilla)

History: Free vs Open Source

- Until Netscape/Mozilla, much of open source movement was **concentrated** in the Free Software Foundation and its GPL

History: Free vs Open Source

- Until Netscape/Mozilla, much of open source movement was **concentrated** in the Free Software Foundation and its GPL
- "**Open Source**" coined in 1998 by the Open Source Initiative as a term to capture Netscape's aim for an **open development process**, Eric Raymond's "Bazaar"

History: Free vs Open Source

- Until Netscape/Mozilla, much of open source movement was concentrated in the Free Software Foundation and its GPL
- "**Open Source**" coined in 1998 by the Open Source Initiative as a term to capture Netscape's aim for an **open development process**, Eric Raymond's "Bazaar"
  - Publisher Tim O'Reilly organizes a "Freeware Summit" later in 1998, soon rebranded as "Open Source Summit"

History: Free vs Open Source

- Until Netscape/Mozilla, much of open source movement was **concentrated** in the Free Software Foundation and its GPL
- “**Open Source**” coined in 1998 by the Open Source Initiative as a term to capture Netscape’s aim for an **open development process**, Eric Raymond’s “Bazaar”
    - 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, FSF founder

Free and Open-source Software

Today’s agenda:

- Finish static analysis slides
- Reading Quiz
- History + the “free software” philosophy
- **Open-source: licenses and business models**
- Mid-semester survey: how am I doing?

What makes an open source project successful?

What makes an open source project successful?

- Open source projects thrive when the **community** surrounding them contributes to push the project forwards

What makes an open source project successful?

- Open source projects thrive when the **community** surrounding them contributes to push the project forwards
- Communities form around **collective ownership** (even if it’s only perceived)

What makes an open source project successful?

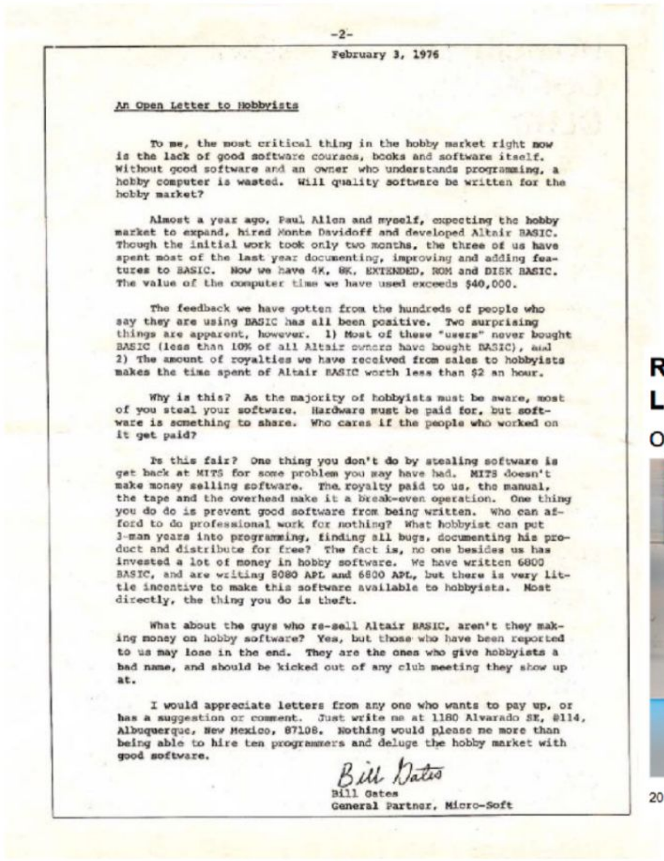
- Open source projects thrive when the **community** surrounding them contributes to push the project forwards
- Communities form around **collective ownership** (even if it’s only perceived)
- Contributors bring **more than code**: also documentation, testing, support, and outreach

What makes an open source project successful?

- Open source projects thrive when the **community** surrounding them contributes to push the project forwards
- Communities form around **collective ownership** (even if it's only perceived)
- Contributors bring **more than code**: also documentation, testing, support, and outreach
- Community/ownership models:
  - Corporate owner, community outreach (MySQL, MongoDB)
  - Foundation owner, corporate sponsors (GNU, Linux)

Is Open Source a Good Business Model?

Is Open Source a Good Business Model?



-2-

February 3, 1976

<u>An Open Letter to Hobbyists</u>

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.

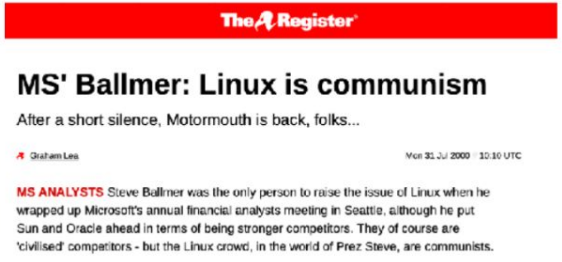
Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

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



The Register

MS' Ballmer: Linux is communism

After a short silence, Motormouth is back, folks...

Graham Lea — Mon 31 Jul 2000 // 10:10 UTC

MS ANALYSTS Steve Ballmer was the only person to raise the issue of Linux when he wrapped up Microsoft's annual financial analysts meeting in Seattle, although he put Sun and Oracle ahead in terms of being stronger competitors. They of course are 'civilised' competitors - but the Linux crowd, in the world of Prez Steve, are communists.

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

The New York Times

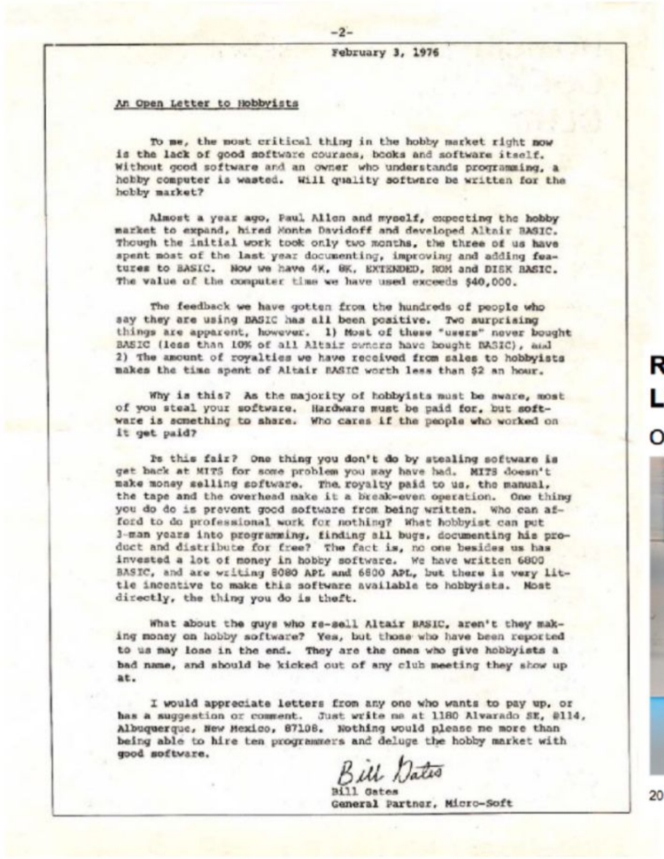
Microsoft Buys GitHub for $7.5 Billion, Moving to Grow in Coding's New Era

Give this article

A GitHub billboard being installed in San Francisco in 2014. Microsoft said on Monday that it would acquire the company for $7.5 billion. David Paul Morris/Bloomberg

By Steve Lohr

Is Open Source a Good Business Model?



-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.

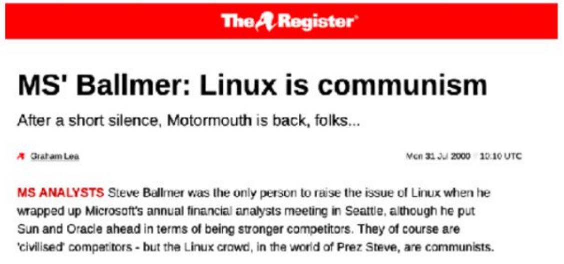
Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

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



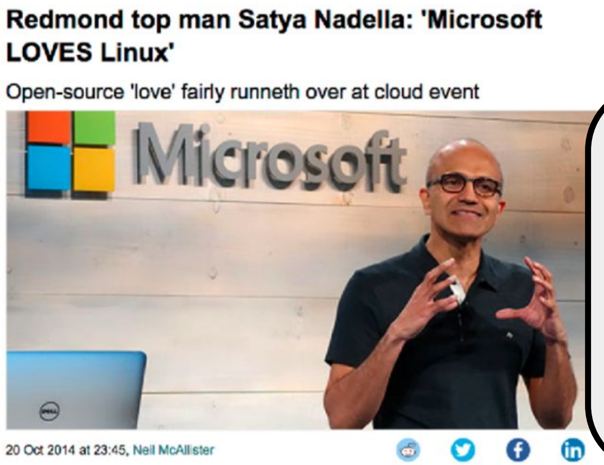
The Register

MS' Ballmer: Linux is communism

After a short silence, Motormouth is back, folks...

Graham Lea — Mon 31 Jul 2000 // 10:10 UTC

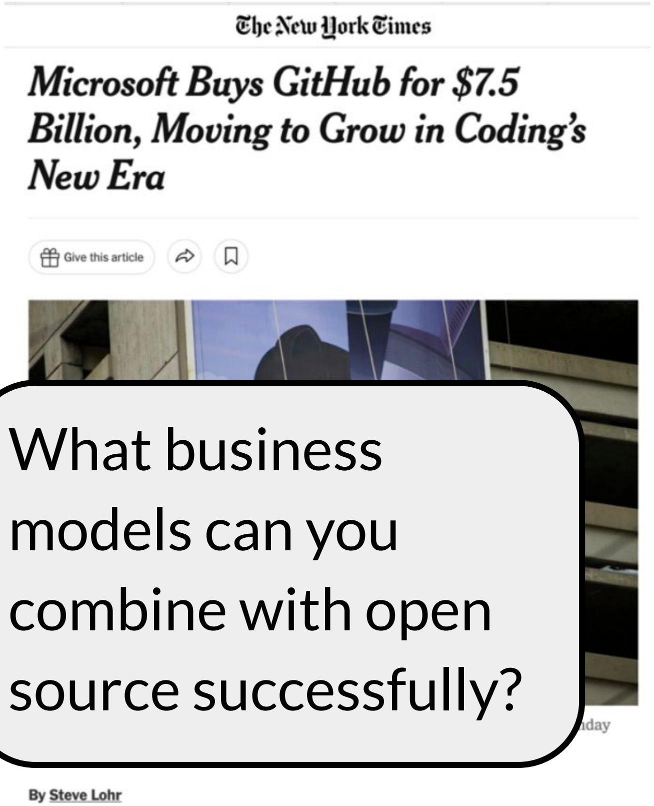
MS ANALYSTS Steve Ballmer was the only person to raise the issue of Linux when he wrapped up Microsoft's annual financial analysts meeting in Seattle, although he put Sun and Oracle ahead in terms of being stronger competitors. They of course are 'civilised' competitors - but the Linux crowd, in the world of Prez Steve, are communists.



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



The New York Times

Microsoft Buys GitHub for $7.5 Billion, Moving to Grow in Coding's New Era

Give this article

nday

By Steve Lohr

What business models can you combine with open source successfully?

Model: “Open Core”, closed plugins

- “**Open Core**” model: core component of a product is an open source utility; **premium plugins** available for a fee

Model: “Open Core”, closed plugins

- “**Open Core**” model: core component of a product is an open source utility; **premium plugins** available for a fee
- Example: Apache Kafka, a distributed message broker (glue in an event-based system)
  - Product is open source, maintained by Apache foundation, supported by company “Confluent”
  - Confluent provides plugins to connect Kafka to many different systems out-of-the-box

Model: Open Source as a Utility

- The largest, most successful open source projects implement **utility infrastructure**:
  - Operating systems, web servers, logging libraries, languages

Model: Open Source as a Utility

- The largest, most successful open source projects implement **utility infrastructure**:
  - Operating systems, web servers, logging libraries, languages
- **Business model**: build and sell products and services using those utilities, contribute improvements back to the ecosystem

Model: Open Source as a Utility

- The largest, most successful open source projects implement **utility infrastructure**:
  - Operating systems, web servers, logging libraries, languages
- **Business model**: build and sell products and services using those utilities, contribute improvements back to the ecosystem
  - i.e., sell **expertise**

Model: Open Source as a Utility

- The largest, most successful open source projects implement **utility infrastructure**:
  - Operating systems, web servers, logging libraries, languages
- **Business model**: build and sell products and services using those utilities, contribute improvements back to the ecosystem
  - i.e., sell **expertise**
  - many companies provide specialized “distributions” of these open source infrastructure and specialized tools to improve them; support the upstream project

Open source and the law

Open source and the law

- **Copyright** provides creators with protection for creative, intellectual and artistic works - **including software**

Open source and the law

- **Copyright** provides creators with protection for creative, intellectual and artistic works - **including software**
    - Alternative: public domain (nobody has exclusive property rights)

Open source and the law

- **Copyright** provides creators with protection for creative, intellectual and artistic works - **including software**
  - Alternative: public domain (nobody has exclusive property rights)
- Open source software is **generally copyrighted**, with copyright retained by contributors or assigned to a foundation/corporation that maintains the product

Open source and the law

- **Copyright** provides creators with protection for creative, intellectual and artistic works - **including software**
  - Alternative: public domain (nobody has exclusive property rights)
- Open source software is **generally copyrighted**, with copyright retained by contributors or assigned to a foundation/corporation that maintains the product
- Copyright holder can grant a ***license*** for use, placing restrictions on how it can be used (perhaps for a fee)
  - Common open source licenses: MIT, BSD, Apache, GPL

Open source licenses

Two broad classes of open source licenses:

Open source licenses

Two broad classes of open source licenses:

- ***permissive licenses*** (e.g., MIT, Apache, BSD) allow a combination of the licensed code and some other code (i.e., a ***derivative work***) to be released under a **different license** (including proprietary)

Open source licenses

Two broad classes of open source licenses:

- *permissive licenses* (e.g., MIT, Apache, BSD) allow a combination of the licensed code and some other code (i.e., a ***derivative work***) to be released under a different license (including proprietary)
  - goal: encourage adoption and use of the software

Open source licenses

Two broad classes of open source licenses:

- *permissive licenses* (e.g., MIT, Apache, BSD) allow a combination of the licensed code and some other code (i.e., a ***derivative work***) to be released under a **different license** (including proprietary)
  - goal: encourage adoption and use of the software
- *copyleft licenses* (e.g., GPL, CC-BY-SA) forces all linked code to be released under the **same license**

Open source licenses

Two broad classes of open source licenses:

- *permissive licenses* (e.g., MIT, Apache, BSD) allow a combination of the licensed code and some other code (i.e., a ***derivative work***) to be released under a **different license** (including proprietary)
  - goal: encourage adoption and use of the software
- *copyleft licenses* (e.g., GPL, CC-BY-SA) forces all linked code to be released under the **same license**
  - goal: protect the commons, require users to contribute back

Open source licenses

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

Two broad classes of open source licenses:

- *permissive licenses* (e.g., MIT, Apache, BSD) allow a combination of the licensed code and some other code (i.e., a ***derivative work***) to be released under a **different license** (including proprietary)
  - goal: encourage adoption and use of the software
- *copyleft licenses* (e.g., GPL, CC-BY-SA) forces all linked code to be released under the **same license**
  - goal: protect the commons, require users to contribute back

Model: Dual Licensing

Model: Dual Licensing

- Offer a **free copyleft** (e.g. GPL) license to encourage broad adoption, prevent competitors from improving it without sharing those improvements.

Model: Dual Licensing

- Offer a **free copyleft** (e.g. GPL) license to encourage broad adoption, prevent competitors from improving it without sharing those improvements.
- Offer **custom, more permissive licenses** to third parties who are willing to pay for that (e.g. enterprise)

Model: Dual Licensing

- Offer a **free copyleft** (e.g. GPL) license to encourage broad adoption, prevent competitors from improving it without sharing those improvements.
- Offer **custom, more permissive licenses** to third parties who are willing to pay for that (e.g. enterprise)
- Only possible when there is a **single copyright owner**, who can unilaterally change license

Model: Dual Licensing

- Offer a **free copyleft** (e.g. GPL) license to encourage broad adoption, prevent competitors from improving it without sharing those improvements.
- Offer **custom, more permissive licenses** to third parties who are willing to pay for that (e.g. enterprise)
- Only possible when there is a **single copyright owner**, who can unilaterally change license
- Risk: losing control of the copyleft portion via **forking**

Model: Dual Licensing

- Offer a **free copyleft** (e.g. GPL) license to encourage broad adoption, prevent competitors from improving it without sharing those improvements.
- Offer **custom, more permissive licenses** to third parties who are willing to pay for that (e.g. enterprise)
- Only possible when there is a **single copyright owner**, who can unilaterally change license
- Risk: losing control of the copyleft portion via **forking**
- Examples: MySQL, Qt

When communities move on: forks

- When software is released under a permissive license, the only rights that the creator can realistically retain are trademarks on name/images - code can otherwise be “***forked***”

When communities move on: forks

- When software is released under a permissive license, the only rights that the creator can realistically retain are trademarks on name/images - code can otherwise be "***forked***"
- Example:
    - Sun bought StarOffice in 1999, GPL open-sourced as OpenOffice in 2000 with aim of fighting MS Office

When communities move on: forks

- When software is released under a permissive license, the only rights that the creator can realistically retain are trademarks on name/images - code can otherwise be “***forked***”
- Example:
    - Sun bought StarOffice in 1999, GPL open-sourced as OpenOffice in 2000 with aim of fighting MS Office
    - 2010: Oracle buys Sun, fires many internal developers, frustrating external community

When communities move on: forks

- When software is released under a permissive license, the only rights that the creator can realistically retain are trademarks on name/images - code can otherwise be "***forked***"
- Example:
  - Sun bought StarOffice in 1999, GPL open-sourced as OpenOffice in 2000 with aim of fighting MS Office
  - 2010: Oracle buys Sun, fires many internal developers, frustrating external community
  - 2011: Community forms a foundation, creates fork LibreOffice, OpenOffice dies off (Oracle transfers to Apache)

Model: Hosted OSS As A Service

Model: Hosted OSS As A Service

- Model: Creators of open source software provide a cloud hosted, "**fully managed**" installation of the software, as a service

Model: Hosted OSS As A Service

- Model: Creators of open source software provide a cloud hosted, "**fully managed**" installation of the software, as a service
- Risk: No competitive advantage vs cloud utility providers (e.g. AWS)

Model: Hosted OSS As A Service

- Model: Creators of open source software provide a cloud hosted, "**fully managed**" installation of the software, as a service
- Risk: No competitive advantage vs cloud utility providers (e.g. AWS)
    - AWS could even improve your GPL code and **not share** because it is **not distributing** the program (it operates it as a service)

Model: Hosted OSS As A Service

- Model: Creators of open source software provide a cloud hosted, "**fully managed**" installation of the software, as a service
- Risk: No competitive advantage vs cloud utility providers (e.g. AWS)
  - AWS could even improve your GPL code and **not share** because it is **not distributing** the program (it operates it as a service)
- Example: MongoDB Atlas (document-oriented database)

Model: Hosted OSS As A Service

- Model: Creators of open source software provide a cloud hosted, "**fully managed**" installation of the software, as a service
- Risk: No competitive advantage vs cloud utility providers (e.g. AWS)
  - AWS could even improve your GPL code and **not share** because it is **not distributing** the program (it operates it as a service)
- Example: MongoDB Atlas (document-oriented database)
  - MongoDB created a **new license** to **require copyleft for service providers** operating MongoDB as a service

Model: Hosted OSS As A Service

- Model: Creators of open source software provide a cloud hosted, "**fully managed**" installation of the software, as a service
- Risk: No competitive advantage vs cloud utility providers (e.g. AWS)
  - AWS could even improve your GPL code and **not share** because it is **not distributing** the program (it operates it as a service)
- Example: MongoDB Atlas (document-oriented database)
  - MongoDB created a **new license** to **require copyleft for service providers** operating MongoDB as a service
  - Amazon created their own fork of the GPL'ed version of MongoDB, ignored code only released under new license

Another example: Java & open-source

- While the Java **specification** is public, there used to be no open source Java runtime **implementation**

Another example: Java & open-source

- While the Java **specification** is public, there used to be no open source Java runtime **implementation**
- Much open source software was/is written in Java, creating "**The Java Trap**" for open source

Another example: Java & open-source

- While the Java **specification** is public, there used to be no open source Java runtime **implementation**
- Much open source software was/is written in Java, creating "**The Java Trap**" for open source
- 1996-2006: GNU, Apache (backed by IBM and Apple), and others attempted to create open source implementations; Sun refused to permit these runtimes to be tested for compatibility, prohibiting them from using the term "Java"

Another example: Java & open-source

- While the Java **specification** is public, there used to be no open source Java runtime **implementation**
- Much open source software was/is written in Java, creating “**The Java Trap**” for open source
- 1996-2006: GNU, Apache (backed by IBM and Apple), and others attempted to create open source implementations; Sun refused to permit these runtimes to be tested for compatibility, prohibiting them from using the term “Java”
- 2007: Sun releases OpenJDK under GPL; third party projects abandoned mostly uncompleted

Another example: Java & ope

- While the Java **specification** is public, t
  source Java runtime **implementation**
- Much open source software was/is written in Java, creating "**The Java Trap**" for open source
- 1996-2006: GNU, Apache (backed by IBM and Apple), and others attempted to create open source implementations; Sun refused to permit these runtimes to be tested for compatibility, prohibiting them from using the term "Java"
- 2007: Sun releases OpenJDK under GPL; third party projects abandoned mostly uncompleted

Why did Sun release OpenJDK?

They feared losing control of Java.

Another example: Java & ope

- While the Java **specification** is public, t
  source Java runtime **implementation**
- Much open source software was/is written in Java, creating "**The Java Trap**" for open source
- 1996-2006: GNU, Apache (backed by IBM and Apple), and others attempted to create open source implementations; Sun refused to permit these runtimes to be tested for compatibility, prohibiting them from using the term "Java"
- 2007: Sun releases OpenJDK under GPL; third party projects abandoned mostly uncompleted

Why did Sun release OpenJDK?

They feared **losing control** of Java.

Another example: Android

Another example: Android

- Model: “Product” is the **ecosystem** (app store, ads, etc) and the hardware (made by competing manufacturers), not Android itself

Another example: Android

- Model: “Product” is the **ecosystem** (app store, ads, etc) and the hardware (made by competing manufacturers), not Android itself
- Android is **entirely open source**, built on Linux; applications are written in Java, executed using a custom-built runtime

Another example: Android

- Model: “Product” is the **ecosystem** (app store, ads, etc) and the hardware (made by competing manufacturers), not Android itself
- Android is **entirely open source**, built on Linux; applications are written in Java, executed using a custom-built runtime
- To provide implementations of **core Java APIs** (e.g. java.util.X), Android used the open source Apache Harmony implementations

Another example: Android

- Model: “Product” is the **ecosystem** (app store, ads, etc) and the hardware (made by competing manufacturers), not Android itself
- Android is **entirely open source**, built on Linux; applications are written in Java, executed using a custom-built runtime
- To provide implementations of **core Java APIs** (e.g. java.util.X), Android used the open source Apache Harmony implementations
- Oracle v Google: Oracle asserted that Java APIs were their property (copyright) and Google misused that; judge ruled that **APIs specifications cannot be copyrighted**

Risks of using Open Source in Industry

Risks of using Open Source in Industry

- Are licenses **compatible**? A significant concern for licenses with copyleft:

Risks of using Open Source in Industry

- Are licenses **compatible**? A significant concern for licenses with copyleft:
    - Adopting libraries with copyleft clause generally means what you distribute linked against that library **must** also have same copyleft clause (and be open source)

Risks of using Open Source in Industry

- Are licenses **compatible**? A significant concern for licenses with copyleft:
  - Adopting libraries with copyleft clause generally means what you distribute linked against that library **must** also have same copyleft clause (and be open source)
  - Including permissive-licensed software in copyleft-licensed software is generally compatible

Risks of using Open Source in Industry

- Are licenses **compatible**? A significant concern for licenses with copyleft:
  - Adopting libraries with copyleft clause generally means what you distribute linked against that library **must** also have same copyleft clause (and be open source)
  - Including permissive-licensed software in copyleft-licensed software is generally compatible
- Are you certain that the software truly is released under the license that is stated? Did all contributors agree to that license?

Risks of using Open Source

Industry must balance these risks against the **clear benefit** of OSS: reusing existing code

- Are licenses **compatible**? A significa copyleft:
  - Adopting libraries with copyleft c at you distribute linked against that library **must** also have same copyleft clause (and be open source)
  - Including permissive-licensed software in copyleft-licensed software is generally compatible
- Are you certain that the software truly is released under the license that is stated? Did all contributors agree to that license?

Licensing and Large Language Models (LLMs)

- Recent development: large language models trained on **all code** in public repositories on GitHub (e.g., Codex model)

Licensing and Large Language Models (LLMs)

- Recent development: large language models trained on **all code** in public repositories on GitHub (e.g., Codex model)
- Tools like GitHub Copilot **suggest lines of code** as you program, based on the Codex model

Licensing and Large Language Models (LLMs)

- Recent development: large language models trained on **all code** in public repositories on GitHub (e.g., Codex model)
- Tools like GitHub Copilot **suggest lines of code** as you program, based on the Codex model
    - Copilot has been observed to output **entire snippets** of code from public GitHub repositories

Licensing and Large Language Models (LLMs)

- Recent development: large language models trained on **all code** in public repositories on GitHub (e.g., Codex model)
- Tools like GitHub Copilot **suggest lines of code** as you program, based on the Codex model
  - Copilot has been observed to output **entire snippets** of code from public GitHub repositories
- Ongoing **legal battles** over:

Licensing and Large Language Models (LLMs)

- Recent development: large language models trained on **all code** in public repositories on GitHub (e.g., Codex model)
- Tools like GitHub Copilot **suggest lines of code** as you program, based on the Codex model
  - Copilot has been observed to output **entire snippets** of code from public GitHub repositories
- Ongoing **legal battles** over:
  - Does training Codex on public code **violate copyleft** licenses?

Licensing and Large Language Models (LLMs)

- Recent development: large language models trained on **all code** in public repositories on GitHub (e.g., Codex model)
- Tools like GitHub Copilot **suggest lines of code** as you program, based on the Codex model
  - Copilot has been observed to output **entire snippets** of code from public GitHub repositories
- Ongoing **legal battles** over:
  - Does training Codex on public code **violate copyleft** licenses?
  - Who is the **owner** of Copilot's output, especially when it is similar to public code that has an owner?

Licensing and Large Language Models (LLMs)

- Recent development: large language models trained on **all code** in public repositories on GitHub (e.g., Codex model)
- Tools like GitHub Copilot **sugges**
  based on the Codex model
  - Copilot has been observed to … e
    from public GitHub repositor
- Ongoing **legal battles** over:
  - Does training Codex on public code **violate copyleft** licenses?
  - Who is the **owner** of Copilot's output, especially when it is similar to public code that has an owner?

Many companies **forbid** their developers from using Copilot or similar tools because of the risks from these legal battles!

Takeaways: free and open-source software

- Free software and open-source software represent different **philosophies** about how code should be shared:
  - Free software: if I share with you, you need to share with me
  - Open source software: I share with you, you do what you want
- Because software is copyrightable, licenses enforce philosophy
  - **copyleft** licenses enforce free software principles
- Many viable open source business models, but all have risks
- **Licensing concerns** are the main reason to avoid open-source code in industry (industry loves permissive licenses)

Free and Open-source Software

Today's agenda:

- Finish static analysis slides
- Reading Quiz
- History + the "free software" philosophy
- Open-source: licenses and business models
- **Mid-semester survey: how am I doing?**

Mid-semester survey: anonymous

https://tinyurl.com/3r9j873j