What is Software Engineering?

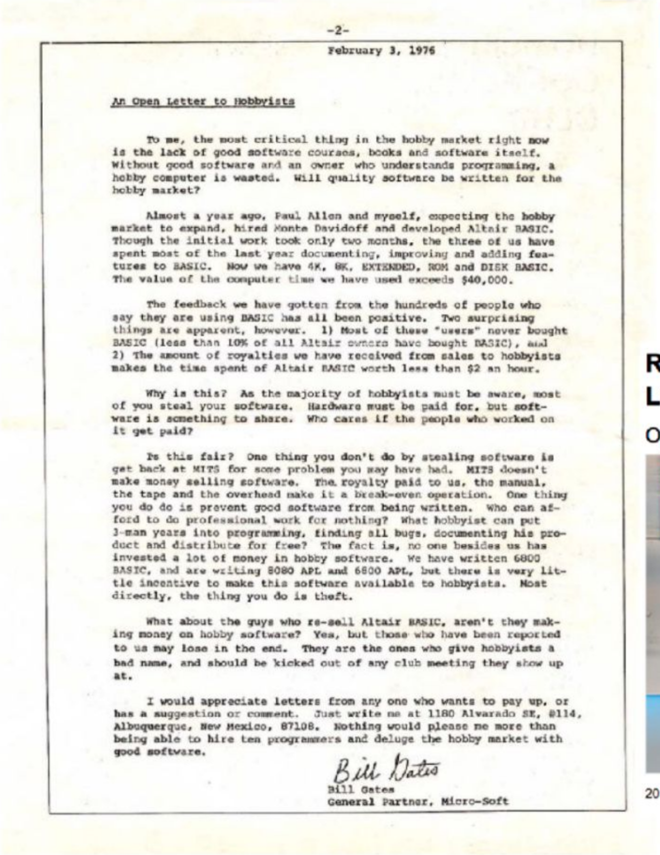
Martin Kellogg

What is Software Engineering?

Today’s agenda:

- **Finish slides from last Friday**
- What is research? How is it similar/different from SE generally?
- Your relationship to researchers, as a developer
- What sort of problems does SE research solve

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.

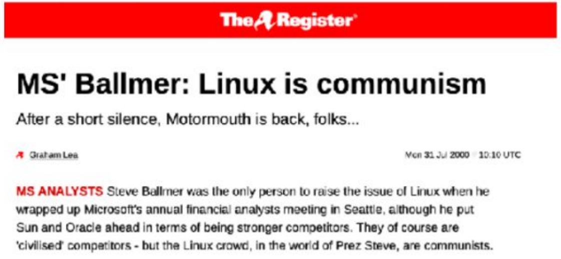
Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

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



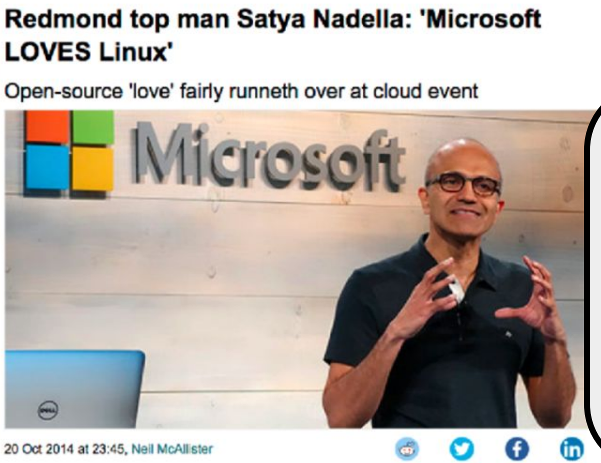
The Register

MS' Ballmer: Linux is communism

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

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

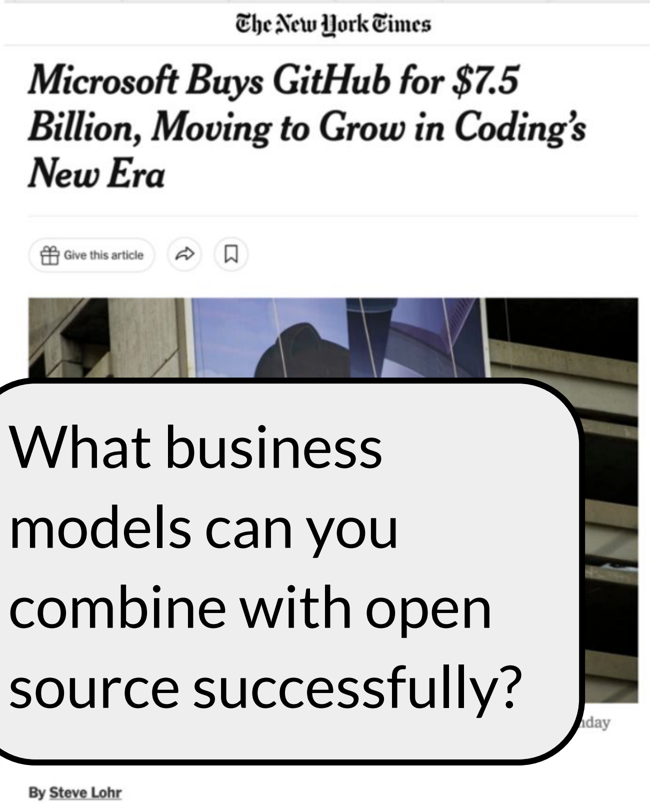
MS ANALYSTS Steve Ballmer was the only person to raise the issue of Linux when he wrapped up Microsoft's annual financial analysts meeting in Seattle, although he put Sun and Oracle ahead in terms of being stronger competitors. They of course are 'civilised' competitors - but the Linux crowd, in the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

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

Why did Sun release OpenJDK?

They feared losing control of Java.

- 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

Another example: Java & ope

Why did Sun release OpenJDK?

They feared losing control of Java.

- 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

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/Kotlin, 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/Kotlin, 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/Kotlin, 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, GPTs)

Licensing and Large Language Models (LLMs)

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

Advice: Large Language Models (LLMs) in SE

Advice: Large Language Models (LLMs) in SE

- Current trends suggest that LLMs are going to be a **major part** of software engineering (and many other disciplines) going forward

Advice: Large Language Models (LLMs) in SE

- Current trends suggest that LLMs are going to be a **major part** of software engineering (and many other disciplines) going forward
    - many engineers **want** to use them, even if they're not currently permitted to due to legal risks
        - great for generating boilerplate, tests, etc.

Advice: Large Language Models (LLMs) in SE

- Current trends suggest that LLMs are going to be a **major part** of software engineering (and many other disciplines) going forward
    - many engineers **want** to use them, even if they're not currently permitted to due to legal risks
        - great for generating boilerplate, tests, etc.
- My view: LLMs are like an **untrustworthy but very smart compiler**

Advice: Large Language Models (LLMs) in SE

- Current trends suggest that LLMs are going to be a **major part** of software engineering (and many other disciplines) going forward
    - many engineers **want** to use them, even if they're not currently permitted to due to legal risks
        - great for generating boilerplate, tests, etc.
- My view: LLMs are like an **untrustworthy but very smart compiler**
    - unlike traditional compiler, do not promise to preserve semantics (and might **hallucinate**)

Advice: Large Language Models (LLMs) in SE

- Current trends suggest that LLMs are going to be a **major part** of software engineering (and many other disciplines) going forward
  - many engineers **want** to use them, even if they're not currently permitted to due to legal risks
    - great for generating boilerplate, tests, etc.
- My view: LLMs are like an **untrustworthy but very smart compiler**
  - unlike traditional compiler, do not promise to preserve semantics (and might **hallucinate**)
  - but input can be natural language or a specification, rather than another program

Advice: Large Language Models (LLMs) in SE

- Current trends suggest that LLMs are going to be a **major part** of software engineering (and rward
  - many engineers **wan** rently permitted to due to l
    - great for generat
- My view: LLMs are like a **mpiler**
  - unlike traditional con semantics (and might **hallucinate**)
  - but input can be natural language or a specification, rather than another program

Possible future workflow:

1. LLMs generate code
2. deductive verification tools check for correctness
3. SDE reviews final code

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)

What is Software Engineering?

Today's agenda:

- Finish slides from last Friday
- **What is research? How is it similar/different from SE generally?**
- Your relationship to researchers, as a developer
- What sort of problems does SE research solve

What is research?

What is research?

- ***Research*** is the process of innovation: creating or discovering something that has never been built/known before

What is research?

- ***Research*** is the process of innovation: creating or discovering something that has never been built/known before
- All software development is to some extent **innovative**

What is research?

- ***Research*** is the process of innovation: creating or discovering something that has never been built/known before
- All software development is to some extent **innovative**
  - the cost of copying software is zero, so any new software has **by definition** not been created before

What is research?

- ***Research*** is the process of innovation: creating or discovering something that has never been built/known before
- All software development is to some extent **innovative**
  - the cost of copying software is zero, so any new software has **by definition** not been created before
  - this contrasts with many other fields, where practitioners ("engineers" or otherwise) are **not** doing anything fundamentally novel

What is research?

- ***Research*** is the process of innovation: creating or discovering something that has never been built/known before
- All software development is to some extent **innovative**
    - the cost of copying software is zero, so any new software has **by definition** not been created before
    - this contrasts with many other fields, where practitioners ("engineers" or otherwise) are **not** doing anything fundamentally novel
        - in those field, anyone doing something new is doing "research"

What is research?

- If all software development is innovative, what distinguishes **computer science research** from just doing software engineering?

What is research?

- If all software development is innovative, what distinguishes **computer science research** from just doing software engineering?
  - the key difference is that most computer science research is **meta** in some way

What is research?

- If all software development is innovative, what distinguishes computer science research from just doing software engineering?
  - the key difference is that most computer science research is **meta** in some way
    - e.g., it might explore how to build **classes** of programs, like operating systems (OS) or compilers (PL)

What is research?

- If all software development is innovative, what distinguishes **computer science research** from just doing software engineering?
    - the key difference is that most computer science research is **meta** in some way
        - e.g., it might explore how to build **classes** of programs, like operating systems (OS) or compilers (PL)
        - or, it might explore **foundational notions** of what computers can and cannot do (CS theory)

What is research?

- If all software development is innovative, what distinguishes **computer science research** from just doing software engineering?
  - the key difference is that most computer science research is **meta** in some way
    - e.g., it might explore how to build **classes** of programs, like operating systems (OS) or compilers (PL)
    - or, it might explore **foundational notions** of what computers can and cannot do (CS theory)
    - or explore what computers we can **physically build** (arch)

What is research?

- So then what’s meta about **software engineering** research?

What is research?

- So then what’s meta about **software engineering** research?
- Software engineering researchers study:

What is research?

- So then what’s meta about **software engineering** research?
- Software engineering researchers study:
    - **what** developers do
        - e.g., studies of developers, what makes them more or less productive, etc.

What is research?

- So then what's meta about **software engineering** research?
- Software engineering researchers study:
    - **what** developers do
        - e.g., studies of developers, what makes them more or less productive, etc.
    - **how** they do it
        - e.g., software architecture, design patterns

What is research?

- So then what’s meta about **software engineering** research?
- Software engineering researchers study:
  - **what** developers do
    - e.g., studies of developers, what makes them more or less productive, etc.
  - **how** they do it
    - e.g., software architecture, design patterns
  - better ways to improve **software quality**
    - e.g., new kinds of testing, static analysis, etc.

What is research?

- So then what's meta about **software engineering** research?
- Software engineering researchers study:
  - **what** developers do
    - e.g., studies of developers, what makes them more or less productive, etc.
  - **how** they do it
    - e.g., software architecture, design patterns
  - better ways to improve **software quality**
    - e.g., new kinds of testing, static analysis, etc.
  - and anything else related to improving **developer productivity**

What is research?

We'll come back to this stuff later in the lecture in a bit more detail, with some examples.

- So then what's meta abou
- Software engineering rese
  - what developers do
    - e.g., studies of developers, what makes them more or less productive, etc.
  - how they do it
    - e.g., software architecture, design patterns
  - better ways to improve software quality
    - e.g., new kinds of testing, static analysis, etc.
  - and anything else related to improving developer productivity

Who does research?

Who does research?

- Most computer science research occurs in **universities**
  - including NJIT!

Who does research?

- Most computer science research occurs in **universities**
  - including NJIT!
- Most research is actually done by **students** (especially **PhD students**), working under a professor

Who does research?

- Most computer science research occurs in **universities**
  - including NJIT!
- Most research is actually done by **students** (especially **PhD students**), working under a professor
  - professor supplies high-level research vision + experience and training

Who does research?

- Most computer science research occurs in **universities**
  - including NJIT!
- Most research is actually done by **students** (especially **PhD students**), working under a professor
  - professor supplies high-level research vision + experience and training
  - student does the grunt work of writing code, gather data, etc.

Who does research?

Not just PhD students: as an **undergraduate** you can get involved in research too (I did!)

- Most computer science researc
  - including NJIT!
- Most research is actually done by **students** (especially **PhD students**), working under a professor
  - professor supplies high-level research vision + experience and training
  - student does the grunt work of writing code, gather data, etc.

Who does research?

- Most computer science research occurs in **universities**
  - including NJIT!
- Most research is actually done by **students** (especially **PhD students**), working under a professor
  - professor supplies high-level research vision + experience and training
  - student does the grunt work of writing code, gather data, etc.
- Some research is done in industry

Who does research?

- Most computer science research occurs in **universities**
  - including NJIT!
- Most research is actually done by **students** (especially **PhD students**), working under a professor
  - professor supplies high-level research vision + experience and training
  - student does the grunt work of writing code, gather data, etc.
- Some research is done in industry
  - e.g., Microsoft has MSR, AWS has ARG, etc.

Who does research?

- Most computer science research occurs in **universities**
  - including NJIT!
- Most research is actually done by **students** (especially **PhD students**), working under a professor
  - professor supplies high-level research vision + experience and training
  - student does the grunt work of writing code, gather data, etc.
- Some research is done in industry
  - e.g., Microsoft has MSR, AWS has ARG, etc.
  - sometimes developers do research by accident, too!

Who does research?

- Most computer science research occurs in **universities**
  - including NJIT!
- Most research is ac
  students), working
  - professor suppl ce
    and training
  - student does the grunt work of writing code, gather data, etc.
- Some research is done in industry
  - e.g., Microsoft has MSR, AWS has ARG, etc.
  - sometimes developers do research by accident, too!

However, developers rarely **publish** their research, which is important if you want it to be a part of the **total sum of human knowledge**.

Aside: should you do a PhD?

Aside: should you do a PhD?

- In my experience, most undergrads think that doing a PhD is just like "**more school**".

Aside: should you do a PhD?

- In my experience, most undergrads think that doing a PhD is just like "**more school**".
  - This is a long way from the truth: being a PhD student is more like a **job** that gives you a PhD when you do it long enough

Aside: should you do a PhD?

- In my experience, most undergrads think that doing a PhD is just like “**more school**”.
    - This is a long way from the truth: being a PhD student is more like a **job** that gives you a PhD when you do it long enough
        - for example, PhD students in CS are typically **paid**, although not very much (“stipends”)

Aside: should you do a PhD?

- In my experience, most undergrads think that doing a PhD is just like “more school”.
  - This is a long way from the truth: being a PhD student is more like a **job** that gives you a PhD when you do it long enough
    - for example, PhD students in CS are typically **paid**, although not very much (“stipends”)
    - the PhD student’s **advisor** (a professor) is their boss

Aside: should you do a PhD?

- In my experience, most und like "more school".
    - This is a long way from like a **job** that gives you
        - for example, PhD students in CS are typically **paid**, although not very much ("stipends")
        - the PhD student's **advisor** (a professor) is their boss

Another misconception: in the US, you usually **do not** need a master's degree to start a PhD program!

Aside: should you do a PhD?

- In my experience, most undergrads think that doing a PhD is just like “**more school**”.
    - This is a long way from the truth: being a PhD student is more like a **job** that gives you a PhD when you do it long enough
        - for example, PhD students in CS are typically **paid**, although not very much (“stipends”)
        - the PhD student’s **advisor** (a professor) is their boss
- For this reason, in my opinion more undergraduates should at least **consider** doing a PhD

Aside: should you do a PhD?

- In my experience, most undergrads think that doing a PhD is just like "**more school**".
  - This is a long way from the truth: being a PhD student is more like a **job** that gives you a PhD when you do it long enough
    - for example, PhD students in CS are typically **paid**, although not very much ("stipends")
    - the PhD student's **advisor** (a professor) is their boss
- For this reason, in my opinion more undergraduates should at least **consider** doing a PhD
  - it might be more affordable than you think!

Aside: should you do a PhD?

- Pros of doing a PhD:

Aside: should you do a PhD?

- Pros of doing a PhD:
  - you become a **world expert** in a topic

Aside: should you do a PhD?

- Pros of doing a PhD:
  - you become a **world expert** in a topic
  - push forth the **bounds of human knowledge**

Aside: should you do a PhD?

- Pros of doing a PhD:
  - you become a **world expert** in a topic
  - push forth the **bounds of human knowledge**
  - some jobs are **only accessible** to people with PhDs:

Aside: should you do a PhD?

- Pros of doing a PhD:
  - you become a **world expert** in a topic
  - push forth the **bounds of human knowledge**
  - some jobs are **only accessible** to people with PhDs:
    - professor
      - although you can **teach** without a PhD, you can’t get tenure without one

Aside: should you do a PhD?

- Pros of doing a PhD:
    - you become a **world expert** in a topic
    - push forth the **bounds of human knowledge**
    - some jobs are **only accessible** to people with PhDs:
        - professor
            - although you can **teach** without a PhD, you can't get tenure without one
        - industrial researcher
            - e.g., static analysis designer, ML architecture developer, etc.

Aside: should you do a PhD?

- Cons of doing a PhD:

Aside: should you do a PhD?

- Cons of doing a PhD:
    - it’s a **bad financial decision** (due to opportunity cost)
        - PhD students get paid, but much less than e.g., software engineer salaries

Aside: should you do a PhD?

- Cons of doing a PhD:
    - it's a **bad financial decision** (due to opportunity cost)
        - PhD students get paid, but much less than e.g., software engineer salaries
    - it takes a **long time**
        - typically 4 to 6 years, sometimes longer

Aside: should you do a PhD?

- Cons of doing a PhD:
    - it's a **bad financial decision** (due to opportunity cost)
        - PhD students get paid, but much less than e.g., software engineer salaries
    - it takes a **long time**
        - typically 4 to 6 years, sometimes longer
    - it's **mentally taxing**
        - you're working on only one thing for 4-6 years!
        - rates of mental health problems among PhD students are much higher than the general population

Aside: should you do a PhD?

- If despite those cons, you think a PhD is something you might be interested in, come **talk to me** (or another professor in the department)

Aside: should you do a PhD?

- If despite those cons, you think a PhD is something you might be interested in, come **talk to me** (or another professor in the department)

Which professor to approach? Choose a **research professor** whose work sounds interesting to you (or who you know already from class).

- at NJIT, research professors all have "professor" in the title
- teaching professors are "lecturers"

Aside: should you do a PhD?

- If despite those cons, you think a PhD is something you might be interested in, come **talk to me** (or another professor in the department)

Which professor to approach? Choose a **research professor** whose **work** sounds interesting to you (or who you know already from class).

- at NJIT, research professors all have "professor" in their title
- teaching professors are "lecturers"

to find out about a professor's work, google "their name NJIT" and read their website

Aside: should you do a PhD?

- If despite those cons, you think a PhD is something you might be interested in, come **talk to me** (or another professor in the department)

Which professor to approach? Choose a **research professor** whose work sounds interesting to you (or who you know already from class).

- at NJIT, research professors all have “professor” in the title
- teaching professors are “lecturers”

Aside: should you do a PhD?

- If despite those cons, you think a PhD is something you might be interested in, come **talk to me** (or another professor in the department)
    - high-quality PhD programs require **letters of recommendation** from professors you've worked with, so you should work with a professor :)

Aside: should you do a PhD?

- If despite those cons, you think a PhD is something you might be interested in, come **talk to me** (or another professor in the department)
  - high-quality PhD programs require **letters of recommendation** from professors you've worked with, so you should work with a professor :)
  - it's best to approach professors about joining their research group when you're a **sophomore or junior**

Aside: should you do a PhD?

- If despite those cons, you think a PhD is something you might be interested in, come **talk to me** (or another professor in the department)
    - high-quality PhD programs require **letters of recommendation** from professors you've worked with, so you should work with a professor :)
    - it's best to approach professors about joining their research group when you're a **sophomore or junior**
        - at this stage, you know enough to be useful, but you'll be around long enough that you can ramp up on a project

What is Software Engineering?

Today’s agenda:

- Finish slides from last Friday
- What is research? How is it similar/different from SE generally?
- **Your relationship to researchers, as a developer**
- What sort of problems does SE research solve

Research to a developer

- Assuming you’re not going to do a PhD, why should you care about research in software engineering (or CS in general)?

Research to a developer

- Assuming you’re not going to do a PhD, why should you care about research in software engineering (or CS in general)?
  - CS is a very **fast-changing**, young field
    - implying best practices change a lot: what we’ve covered in 490 might not be true anymore in 5/10/20 years

Research to a developer

- Assuming you’re not going to do a PhD, why should you care about research in software engineering (or CS in general)?
    - CS is a very **fast-changing**, young field
        - implying best practices change a lot: what we’ve covered in 490 might not be true anymore in 5/10/20 years
    - Many developers are also working in fast-changing **domains** within CS
        - e.g., if you’re working on ML, you’ll want to keep up with the latest ML research

Research to a developer

- You may also have **industrial researchers** embedded in your company

Research to a developer

- You may also have **industrial researchers** embedded in your company
  - if you're at a "big tech" company, you definitely do; other places, it's a maybe

Research to a developer

- You may also have **industrial researchers** embedded in your company
  - if you're at a "big tech" company, you definitely do; other places, it's a maybe
- Especially if you're working on something **cutting edge** and you're considering trying to keep up with the latest research yourself, finding an industrial researcher in your company is a good idea
  - they can keep up with the research so you don't have to!

Keeping up with research

Keeping up with research

- **Industry-focused** academic publications
    - e.g., CACM (“Communications of the ACM”) is great for this

Keeping up with research

- **Industry-focused** academic publications
  - e.g., CACM ("Communications of the ACM") is great for this
- Find some **technology bloggers** that you like
  - common tech blog entry: a review of a recent paper by the blogger (they read it so you don't have to!)

Keeping up with research

- **Industry-focused** academic publications
  - e.g., CACM (“Communications of the ACM”) is great for this
- Find some **technology bloggers** that you like
  - common tech blog entry: a review of a recent paper by the blogger (they read it so you don’t have to!)
- Attend **industry conferences** (at your employer’s expense…)

Keeping up with research

- **Industry-focused** academic publications
  - e.g., CACM ("Communications of the ACM") is great for this
- Find some **technology bloggers** that you like
  - common tech blog entry: a review of a recent paper by the blogger (they read it so you don't have to!)
- Attend **industry conferences** (at your employer's expense…)
- Keep up with research areas you're particularly interested in directly, by reading (or, more likely, **skimming**) papers
  - more advice on this next

Reading papers

- I strongly recommend that you **skim** papers as a developer

Reading papers

- I strongly recommend that you **skim** papers as a developer
    - if you’re going to read them at all

Reading papers

- I strongly recommend that you **skim** papers as a developer
  - if you’re going to read them at all
- “**skimming**” = “reading only the **most important results**, and skipping the details of how those results were reached”

Reading papers

- I strongly recommend that you **skim** papers as a developer
  - if you’re going to read them at all
- “**skimming**” = “reading only the **most important results**, and skipping the details of how those results were reached”
  - in academic papers, this usually means reading just the abstract and introduction (and maybe the conclusion)

Reading papers

- I strongly recommend that you **skim** papers as a developer
    - if you’re going to read them at all
- “**skimming**” = “reading only the **most important results**, and skipping the details of how those results were reached”
    - in academic papers, this usually means reading just the abstract and introduction (and maybe the conclusion)
- Be careful, though: **not** all academic papers are **equally high-quality**!

Reading papers

- I strongly recommend that you **skim** papers as a developer
  - if you’re going to read them at all
- “**skimming**” = “reading only the **most important results**, and skipping the details of how those results were reached”
  - in academic papers, this usually means reading just the abstract and introduction (and maybe the conclusion)
- Be careful, though: **not** all academic papers are **equally high-quality**!
  - as a dev, you’re not trained to judge this, so relying on peer review + recommendations from e.g., tech bloggers is smart

Reading papers

- I strongly recommend that you **skim** papers as a developer
  - if you're going to read them at all
- "**skimming**" = "reading o
  skipping the details of
  - in academic paper ct
    and introduction (a
- Be careful, though: **no**
  high-quality!
  - as a dev, you're not
    review + recommendations from e.g., tech bloggers is smart

Exception: papers published by **industrial research labs** (e.g., Google Research, MSR) are almost always written in a style closer to what developers are trained to read. These are often the ones you want to focus on as a developer, anyway!

Reading papers: finding papers

Reading papers: finding papers

- In computer science, new research is usually published in **conferences** (not journals)

Reading papers: finding papers

- In computer science, new research is usually published in **conferences** (not journals)
  - conferences have shorter **publication lag**, often < 6 months

Reading papers: finding papers

- In computer science, new research is usually published in **conferences** (not journals)
    - conferences have shorter **publication lag**, often < 6 months
- If you want to get a feel for the latest research in a part of CS, you need to find the **best conferences** for that field
    - usually, fields have many conferences, of which only 2-4 are high-quality

Reading papers: finding papers

- In computer science, new research is usually published in **conferences** (not journals)
  - conferences have shorter **publication lag**, often < 6 months
- If you want to get a feel for the latest research in a part of CS, you need to find the **best conferences** for that field
  - usually, fields have many conferences, of which only 2-4 are high-quality
- To find the best conferences, you could:

Reading papers: finding papers

- In computer science, new research is usually published in **conferences** (not journals)
  - conferences have shorter **publication lag**, often < 6 months
- If you want to get a feel for the latest research in a part of CS, you need to find the **best conferences** for that field
  - usually, fields have many conferences, of which only 2-4 are high-quality
- To find the best conferences, you could:
  - ask a peer in industrial research (if you have one)

Reading papers: finding papers

- In computer science, new research is usually published in **conferences** (not journals)
    - conferences have shorter **publication lag**, often < 6 months
- If you want to get a feel for the latest research in a part of CS, you need to find the **best conferences** for that field
    - usually, fields have many conferences, of which only 2-4 are high-quality
- To find the best conferences, you could:
    - ask a peer in industrial research (if you have one)
    - use a website like csrankings.org

What is Software Engineering?

Today's agenda:

- Finish slides from last Friday
- What is research? How is it similar/different from SE generally?
- Your relationship to researchers, as a developer
- **What sort of problems does SE research solve**

Software Engineering Research

Software Engineering Research

- Some research areas in CS are united by **methodology**
    - e.g., most PL papers are "compilers for X"

Software Engineering Research

- Some research areas in CS are united by **methodology**
  - e.g., most PL papers are “compilers for X”
- Other areas are united by **application**
  - e.g., most OS papers are about operating systems

Software Engineering Research

- Some research areas in CS are united by **methodology**
  - e.g., most PL papers are “compilers for X”
- Other areas are united by **application**
  - e.g., most OS papers are about operating systems
- Software engineering research is united by an application: **developer productivity**

Software Engineering Research

- Some research areas in CS are united by **methodology**
  - e.g., most PL papers are “compilers for X”
- Other areas are united by **application**
  - e.g., most OS papers are about operating systems
- Software engineering research is united by an application: **developer productivity**
  - as a developer, this is an application you will probably care about

Software Engineering Research

- Some research areas in CS are united by **methodology**
  - e.g., most PL papers are "compilers for X"
- Other areas are united by **application**
  - e.g., most OS papers are about operating systems
- Software engineering research is united by an application: **developer productivity**
  - as a developer, this is an application you will probably care about
  - so SE research is particularly important to developers!

Reading quiz

Q1: the author references a paper by Redwine and Riddle repeatedly. That paper is about which of the following topics?

A. program verification
B. automated testing
C. technology maturation
D. software architecture

Q2: **TRUE** or **FALSE**: the author compares software engineering research to (and takes inspiration from) a series of "pro forma" abstracts from the operating systems research community.

Reading quiz

Q1: the author references a paper by Redwine and Riddle repeatedly. That paper is about which of the following topics?

A. program verification

B. automated testing

C. technology maturation

D. software architecture

Q2: **TRUE** or **FALSE**: the author compares software engineering research to (and takes inspiration from) a series of “pro forma” abstracts from the operating systems research community.

Reading quiz

Q1: the author references a paper by Redwine and Riddle repeatedly. That paper is about which of the following topics?

A. program verification

B. automated testing

C. technology maturation

D. software architecture

Q2: **TRUE** or **FALSE**: the author compares software engineering research to (and takes inspiration from) a series of “pro forma” abstracts from the operating systems research community.

Reading quiz

Q1: the author references a paper by Redwine and Riddle repeatedly. That paper is about which of the following topics?

A. program verification
B. automated testing
C. technology maturation
D. software architecture

Q2: **TRUE** or **FALSE**: the author compares software engineering research to (and takes inspiration from) a series of "pro forma" abstracts from the ~~operating systems~~ **HCI** research community.

Wrapup

- I hope you enjoyed CS 490 this semester
- (but we still have one more class: next Wednesday, you have to present to me!)