What is Software Engineering?

Martin Kellogg

Reading quiz: SE + research

Q1: **TRUE** or **FALSE**: the author argues that major results in fields like software engineering gain credibility over time as successive papers provide incremental improvement of the result and progressively stronger credibility.

Q2: When and where is this class' final exam? (Give the date, start time, and room. Hint: it is in one of the two rooms that class is held in, so you just need to unambiguously specify which.)

Reading quiz: SE + research

Q1: **TRUE** or **FALSE**: the author argues that major results in fields like software engineering gain credibility over time as successive papers provide incremental improvement of the result and progressively stronger credibility.

Q2: When and where is this class' final exam? (Give the date, start time, and room. Hint: it is in one of the two rooms that class is held in, so you just need to unambiguously specify which.)

Reading quiz: SE + research

Q1: **TRUE** or **FALSE**: the author argues that major results in fields like software engineering gain credibility over time as successive papers provide incremental improvement of the result and progressively stronger credibility.

Q2: When and where is this class' final exam? (Give the date, start time, and room. Hint: it is in one of the two rooms that class is held in, so you just need to unambiguously specify which.)

Monday, December 16th at 11:30am, in GITC 1100.

Announcements

- Exam review session will be on **Friday evening**
  - 5-6:30pm on Zoom; I will post the link on Discord
  - bring questions; ends early if there are no more questions
- Extra OH Friday morning 9-10:30 (but no regular OH on Thursday)
- Course evaluations close tonight
  - please fill it out! I do read them…
  - I'll give you ~15-20 minutes at the end of class today (hopefully)
- Final demo attendance is mandatory
  - all demos are in my office (GITC 4314)
  - time slots will be strictly enforced

What is Software Engineering?

Today's agenda:

- **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).

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)

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

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

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:

- 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
  - (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
  skipping the details of
  - in academic paper
    and introduction (
- 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:

- 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 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 probably care about
  - so SE research is particularly important to developers!

What's Hot in Software Engineering Research

- My goal in this section is to give you a **taste** of some of research going on in the software engineering community right now
  - these slides aren't exhaustive

What's Hot in Software Engineering Research

- My goal in this section is to give you a **taste** of some of research going on in the software engineering community right now
  - these slides aren't exhaustive
- If you **want to know more** about any of this, come by my office hours or make an appointment with me - I love to talk about this stuff!

What's Hot: Using LLMs in SE

Applications of LLM and Other AI Technologies

- Liuqing Chen, Yunnong Chen, Shuhong Xiao, Yaxuan Song, Lingyun Sun, Yankun Zhen, Tingting Zhou, Yanfang Chang:
  EGFE: End-to-end Grouping of Fragmented Elements in UI Designs with Multimodal Learning. 11:1-11:12
- Cuiying Gao, Gaozhun Huang, Heng Li, Bang Wu, Yueming Wu, Wei Yuan:
  A Comprehensive Study of Learning-based Android Malware Detectors under Challenging Environments. 12:1-12:13
- Antonio Mastropaolo, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta:
  Toward Automatically Completing GitHub Workflows. 13:1-13:12
- Junjielong Xu, Ziang Cui, Yuan Zhao, Xu Zhang, Shilin He, Pinjia He, Liqun Li, Yu Kang, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, Dongmei Zhang:
  UniLog: Automatic Logging via LLM and In-Context Learning. 14:1-14:12
- Yutong Wang, Cindy Rubio-González:
  Predicting Performance and Accuracy of Mixed-Precision Programs for Precision Tuning. 15:1-15:13
- Benjamin Steenhoek, Hongyang Gao, Wei Le:
  Dataflow Analysis-Inspired Deep Learning for Efficient Vulnerability Detection. 16:1-16:13
- Aidan Z. H. Yang, Claire Le Goues, Ruben Martins, Vincent J. Hellendoorn:
  Large Language Models for Test-Free Fault Localization. 17:1-17:12

What's Hot: Using LLMs in SE

Applications of LLM and Other AI Technologies

Liuqing Chen, Yunnong Chen, Shuhong Xiao, Yaxuan Song, Lingyun Sun, Yankun Zhen, Tingting Zhou, Yanfang Chang:
EGFE: End-to-end Grouping of Fragmented Elements in UI Designs with Multimodal Learning. 11:1-11:12

DNN and Language Models for Code

Binhang Qi, Hailong Sun, Hongyu Zhang, Ruobing Zhao, Xiang Gao:
Modularizing while Training: A New Paradigm for Modularizing DNN Models. 31:1-31:12

Lipeng Ma, Weidong Yang, Bo Xu, Sihang Jiang, Ben Fei, Jiaqing Liang, Mingjie Zhou, Yanghua Xiao:
KnowLog: Knowledge Enhanced Pre-trained Language Model for Log Understanding. 32:1-32:13

Changan Niu, Chuanyi Li, Vincent Ng, David Lo, Bin Luo:
FAIR: Flow Type-Aware Pre-Training of Compiler Intermediate Representations. 33:1-33:12

Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, Xin Peng:
Exploring the Potential of ChatGPT in Automated Code Refinement: An Empirical Study. 34:1-34:13

Boxi Yu, Jiayi Yao, Qiuai Fu, Zhiqing Zhong, Haotian Xie, Yaoliang Wu, Yuchi Ma, Pinjia He:
Deep Learning or Classical Machine Learning? An Empirical Study on Log-Based Anomaly Detection. 35:1-35:13

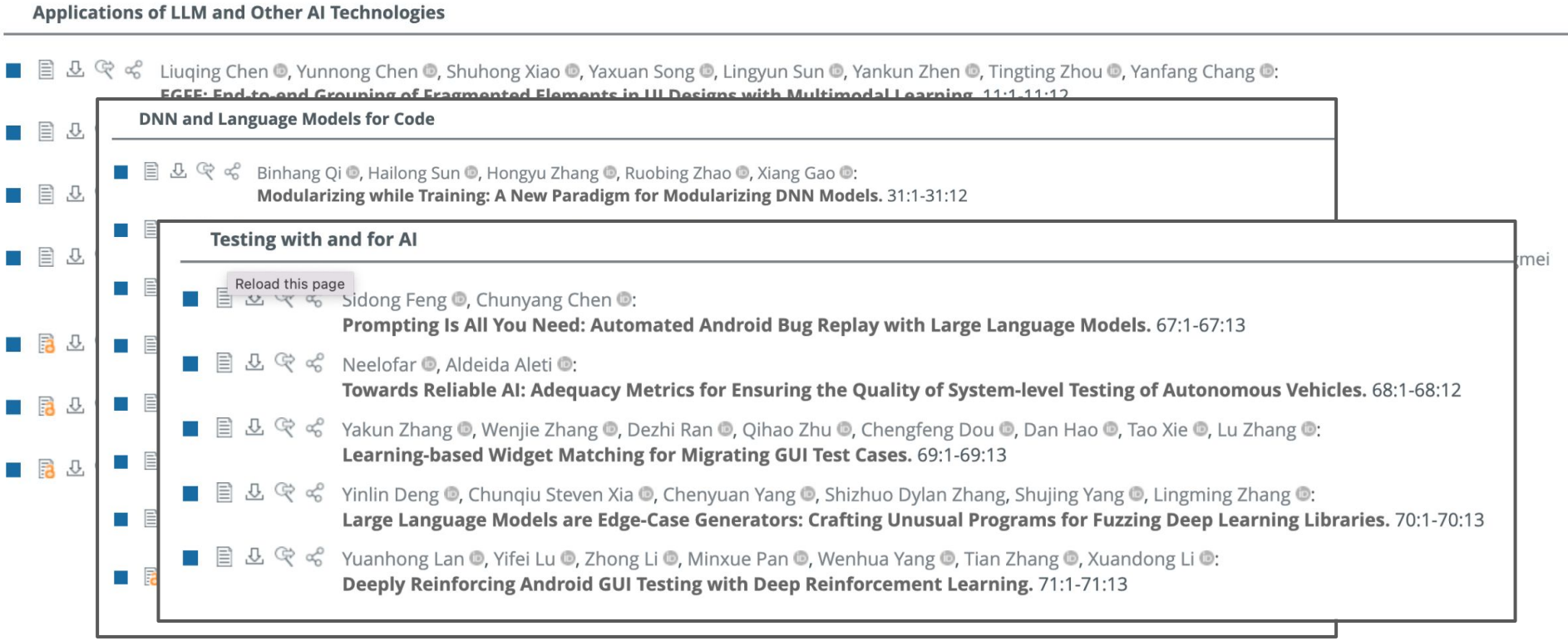
Yangruibo Ding, Benjamin Steenhoek, Kexin Pei, Gail E. Kaiser, Wei Le, Baishakhi Ray:
TRACED: Execution-aware Pre-training for Source Code. 36:1-36:12

Hao Yu, Bo Shen, Dezhi Ran, Jiaxin Zhang, Qi Zhang, Yuchi Ma, Guangtai Liang, Ying Li, Qianxiang Wang, Tao Xie:
CoderEval: A Benchmark of Pragmatic Code Generation with Generative Pre-trained Models. 37:1-37:12

Shibbir Ahmed, Hongyang Gao, Hridesh Rajan:
Inferring Data Preconditions from Deep Learning Models for Trustworthy Prediction in Deployment. 38:1-38:13

an Rajmohan, Dongmei

What's Hot: Using LLMs in SE



What's Hot: Using LLMs in SE

All ICSE 2024

Applications of LLM and Other AI Technologies

Liuqing Chen, Yunnong Chen, Shuhong Xiao, Yaxuan Song, Lingyun Sun, Yankun Zhen, Tingting Zhou, Yanfang Chang:
EGFE: End-to-end Grouping of Fragmented Elements in UI Designs with Multimodal Learning. 11:1-11:12

DNN and Language Models for Code

Binhang Qi, Hailong Sun, Hongyu Zhang, Ruobing Zhao, Xiang Gao:
Modularizing while Training: A New Paradigm for Modularizing DNN Models. 31:1-31:12

Testing with and for AI

Reload this page

Sidong Feng, Chunyang Chen:
Prompting Is All You Need: Automated Android Bug Replay with Large Language Models. 67:1-67:13

Neelofar, Aldeida Aleti:
Towards Reliable AI: Adequacy Metrics for Ensuring the Quality of System-level Testing of Autonomous Vehicles. 68:1-68:12

Yakun Zhang, Wenjie Zhang, Dezhi Ran, Qihao Zhu, Chengfeng Dou, Dan Hao, Tao Xie, Lu Zhang:
Learning-based Widget Matching for Migrating GUI Test Cases. 69:1-69:13

Yinlin Deng, Chunqiu Steven Xia, Chenyuan Yang, Shizhuo Dylan Zhang, Shujing Yang, Lingming Zhang:
Large Language Models are Edge-Case Generators: Crafting Unusual Programs for Fuzzing Deep Learning Libraries. 70:1-70:13

Yuanhong Lan, Yifei Lu, Zhong Li, Minxue Pan, Wenhua Yang, Tian Zhang, Xuandong Li:
Deeply Reinforcing Android GUI Testing with Deep Reinforcement Learning. 71:1-71:13

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 [illegible]rward
  - many engineers **wan**[illegible]rently permitted to due to l[illegible]
    - great for generat[illegible]
- My view: LLMs are like a[illegible]**mpiler**
  - unlike traditional com[illegible] 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

What's Hot: Automated Program Repair

AI&SE Program Repair

- Julian Aron Prenner, Romain Robbes:
  Out of Context: How important is Local Context in Neural Program Repair? 83:1-83:13
- Hadeel Eladawy, Claire Le Goues, Yuriy Brun:
  Automated Program Repair, What Is It Good For? Not Absolutely Nothing! 84:1-84:13
- Wenzhang Yang, Linhai Song, Yinxing Xue:
  Rust-lancet: Automated Ownership-Rule-Violation Fixing with Behavior Preservation. 85:1-85:13
- Fairuz Nawer Meem, Justin Smith, Brittany Johnson:
  Exploring Experiences with Automated Program Repair in Practice. 86:1-86:11
- Yiu Wai Chow, Luca Di Grazia, Michael Pradel:
  PyTy: Repairing Static Type Errors in Python. 87:1-87:13
- Xin Zhou, Kisub Kim, Bowen Xu, DongGyun Han, David Lo:
  Out of Sight, Out of Mind: Better Automatic Vulnerability Repair by Broadening Input Ranges and Sources. 88:1-88:13
- Changhua Luo, Wei Meng, Shuai Wang:
  Strengthening Supply Chain Security with Fine-grained Safe Patch Identification. 89:1-89:12
- Shaoheng Cao, Minxue Pan, Yu Pei, Wenhua Yang, Tian Zhang, Linzhang Wang, Xuandong Li:
  Comprehensive Semantic Repair of Obsolete GUI Test Scripts for Mobile Applications. 90:1-90:13
- Zunchen Huang, Chao Wang:
  Constraint Based Program Repair for Persistent Memory Bugs. 91:1-91:12

What's Hot: Automated Program Repair

- Basic ***automated program repair*** (APR) idea:
  - given a test suite with one failing test and the program source
  - make some change so that the test passes

What's Hot: Automated Program Repair

- Basic ***automated program repair*** (APR) idea:
  - given a test suite with one failing test and the program source
  - make some change so that the test passes
- Modern APR revival is based on **promise of LLMs**

What's Hot: Automated Program Repair

- Basic ***automated program repair*** (APR) idea:
  - given a test suite with one failing test and the program source
  - make some change so that the test passes
- Modern APR revival is based on **promise of LLMs**
- But we've been here before…
  - back in 2012, we had APR systems claiming ~50% repair rate

What's Hot: Automated Program Repair

- Basic ***automated program repair*** (APR) idea:
  - given a test suite with one failing test and the program source
  - make some change so that the test passes
- Modern APR revival is based on **promise of LLMs**
- But we've been here before…
  - back in 2012, we had APR systems claiming ~50% repair rate
  - this was mostly hype + bad measurements
    - ask me for more details…
  - maybe this time will be different?

What's Hot: Fuzzing

Fuzzing and Symbolic Execution

Yue Sun, Guowei Yang, Shichao Lv, Zhi Li, Limin Sun:
Concrete Constraint Guided Symbolic Execution. 122:1-122:12

Luiz Carvalho, Renzo Degiovanni, Maxime Cordy, Nazareno Aguirre, Yves Le Traon, Mike Papadakis:
SpecBCFuzz: Fuzzing LTL Solvers with Boundary Conditions. 123:1-123:13

Zhiwu Xu, Bohao Wu, Cheng Wen, Bin Zhang, Shengchao Qin, Mengda He:
RPG: Rust Library Fuzzing with Pool-based Fuzz Target Generation and Generic Support. 124:1-124:13

Xindi Zhang, Bohan Li, Shaowei Cai:
Deep Combination of CDCL(T) and Local Search for Satisfiability Modulo Non-Linear Integer Arithmetic Theory. 125:1-125:13

Chunqiu Steven Xia, Matteo Paltenghi, Jia Le Tian, Michael Pradel, Lingming Zhang:
Fuzz4All: Universal Fuzzing with Large Language Models. 126:1-126:13

Shuohan Wu, Zihao Li, Luyi Yan, Weimin Chen, Muhui Jiang, Chenxu Wang, Xiapu Luo, Hao Zhou:
Are We There Yet? Unraveling the State-of-the-Art Smart Contract Fuzzers. 127:1-127:13

Shuo Yang, Jiachi Chen, Mingyuan Huang, Zibin Zheng, Yuan Huang:
Uncover the Premeditated Attacks: Detecting Exploitable Reentrancy Vulnerabilities by Identifying Attacker Contracts. 128:1-128:12

Katherine Hough, Jonathan Bell:
Crossover in Parametric Fuzzing. 129:1-129:12

Shengcheng Yu, Chunrong Fang, Mingzhe Du, Yuchen Ling, Zhenyu Chen, Zhendong Su:
Practical Non-Intrusive GUI Exploration Testing with Visual-based Robotic Arms. 130:1-130:13

Xuwei Liu, Wei You, Yapeng Ye, Zhuo Zhang, Jianjun Huang, Xiangyu Zhang:
FuzzInMem: Fuzzing Programs via In-memory Structures. 131:1-131:13

Danushka Liyanage, Seongmin Lee, Chakkrit Tantithamthavorn, Marcel Böhme:
Extrapolating Coverage Rate in Greybox Fuzzing. 132:1-132:12

What's Hot: Fuzzing

Fuzzing and Symbolic Execution

Yue Sun, Guowei Yang, Shichao Lv, Zhi Li, Limin Sun:
Concrete Constraint Guided Symbolic Execution. 122:1-122:12

Luiz Carvalho, Renzo Degiovanni, Maxime Cordy, Nazareno Aguirre, Yves Le Traon, Mike Papadakis:
SpecBCFuzz: Fuzzing LTL Solvers with Boundary Conditions. 123:1-123:13

Zhiw
RPG:

Xind
Deep

Chur
Fuzz

Shuc
Are

Shuc
Unc

Kath
Cros

Sher
Prac

Xuw
Fuzz

Danu
Extr

Fuzzing and Vulnerability Detection

Junda He, Zhou Yang, Jieke Shi, Chengran Yang, Kisub Kim, Bowen Xu, Xin Zhou, David Lo:
Curiosity-Driven Testing for Sequential Decision-Making Process. 165:1-165:14

Yuqiang Sun, Daoyuan Wu, Yue Xue, Han Liu, Haijun Wang, Zhengzi Xu, Xiaofei Xie, Yang Liu:
GPTScan: Detecting Logic Vulnerabilities in Smart Contracts by Combining GPT with Program Analysis. 166:1-166:13

Qi Zhan, Xing Hu, Zhiyang Li, Xin Xia, David Lo, Shanping Li:
PS3: Precise Patch Presence Test based on Semantic Symbolic Signature. 167:1-167:12

Zhijie Zhong, Zibin Zheng, Hong-Ning Dai, Qing Xue, Junjia Chen, Yuhong Nan:
PrettySmart: Detecting Permission Re-delegation Vulnerability for Token Behaviors in Smart Contracts. 168:1-168:12

Huanting Wang, Zhanyong Tang, Shin Hwei Tan, Jie Wang, Yuzhe Liu, Hejun Fang, Chunwei Xia, Zheng Wang:
Combining Structured Static Code Information and Dynamic Symbolic Traces for Software Vulnerability Prediction. 169:1-169:13

Feng Luo, Ruijie Luo, Ting Chen, Ao Qiao, Zheyuan He, Shuwei Song, Yu Jiang, Sixing Li:
SCVHunter: Smart Contract Vulnerability Detection Based on Heterogeneous Graph Attention Network. 170:1-170:13

Xun Deng, Sidi Mohamed Beillahi, Cyrus Minwalla, Han Du, Andreas G. Veneris, Fan Long:
Safeguarding DeFi Smart Contracts against Oracle Deviations. 171:1-171:12

Wrapup

- If you remember one thing from this class:
  - software engineering is all about trade-offs!
- I hope you enjoyed CS 490 this semester
- Remaining class time: course evaluations
  - I do read them!
  - find it at canvas.njit.edu or blue.njit.edu/blue