CS453: Automated Software Testing

Admins & Introduction

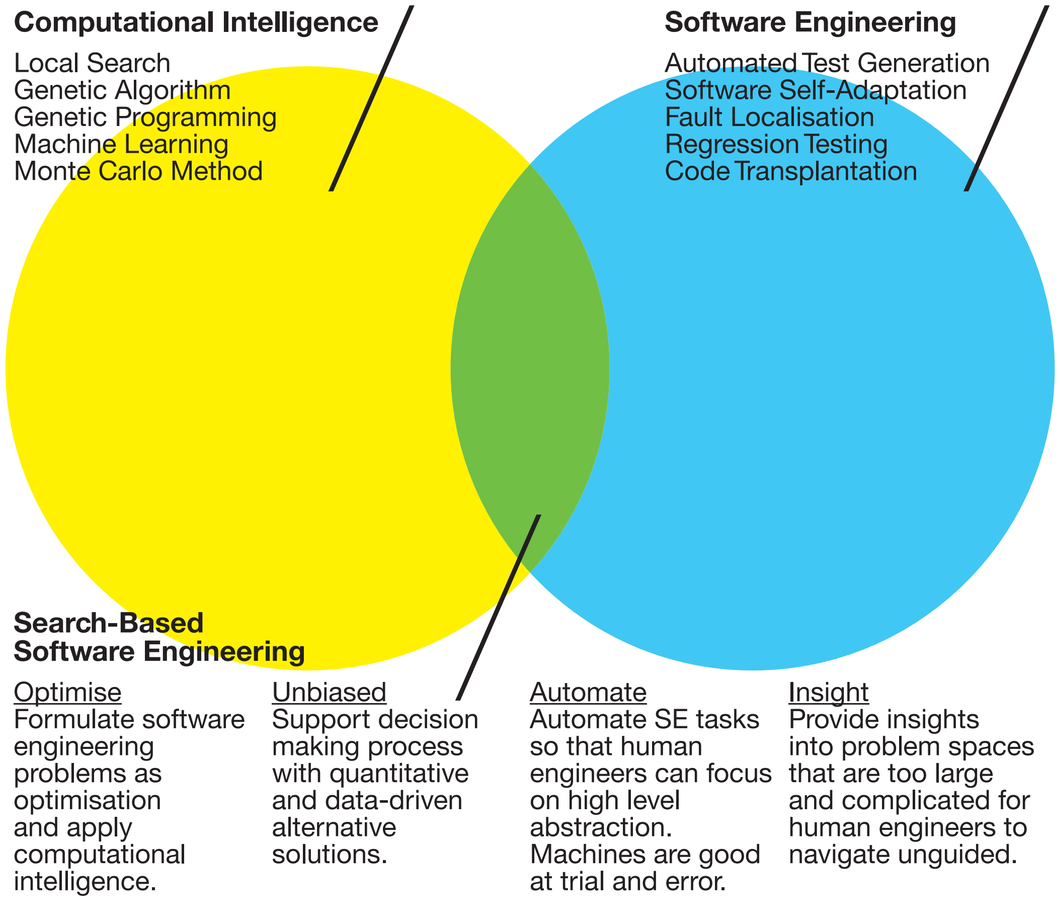
Shin Yoo

COINSE@KAIST

- Associate Professor, KAIST
- Assistant Professor, UCL, UK (2012~2015)
- PhD from King's College London, supervised by Prof. Mark Harman
- Associate Editor at ACM Transactions on Software Engineering and Methodology (TOSEM) Springer Journal of Empirical Software Engineering (EMSE), and Springer Genetic Programming and Evolvable Machines (GPEM)
- IEEE International Conference on Software Testing, Verification & Validation - Steering Committee Chair
- ICSE 2024 Area Co-chair - Testing and Analysis

Computational Intelligence for SE

- At the intersection of machine intelligence and software engineering
- Traditional root in the use of optimisation for SE tasks (i.e., search-based software engineering)
- Gradually adopting more deep learning and NLP into the fold



CS453: Automated Software Testing

- We focus on various concepts and techniques in automated software testing and debugging.
  - We will cover the mainstream software testing techniques: most of them have a heavy emphasis on automation (we will see why).
- With an emphasis on learning how to do meta programming.

Class Communication

- We are trying out Slack workspace as the class communication channel this semester.
- All class announcements, as well as Q&A, will take place on a dedicated workspace: **https://cs453-2024-spring.slack.com**.
- **You must join!** It is strongly recommended that you install either a desktop or a mobile client, to get notifications. Invitation link has been sent in an email from KLMS.
- When you join, use “[Full Name]([STUDENT ID #])” as your username, e.g., “Shin Yoo (20201234)”.
    - #questions for questions, #teams for finding teammates, #random for jokes and memes, #general for anything else

Learning Objectives

- Know fundamental concepts, principles, activities and techniques for software validation and be able to justify appropriate use of techniques for a particular project
- Understand a range of approaches to testing that can be applied to software systems and can apply them appropriately
- Appreciate the limitations of the current tools and have insights in ongoing research topics to overcome them

Schedule

- Please refer to http://coinse.github.io/teaching/2024/cs453
- Spring 2024 courses will be physical.
  - I have pre-committed conference attendances
    - one during mid-term week, so no skipped lectures
    - another potential one in late May (will update the schedule)

Grading

- CS453 gets rid of exams in 2024; instead, we put more focus on implementing stuffs.
  - 60% Assignments (five, each with varying grades)
  - 30% Course Project
  - 10% Quiz

Requirements

- **Strong programming skills**: you are required to actively contribute to group and individual project, which involves serious implementation. There will be also a number of hands-on sessions where we will program together during the class.
- **Unix/Linux-savvy**: you should be familiar with the usual build tools and Unix/Linux command line environments.
- **Git-aware**: you will be required to submit a github repository as part of your project deliverable. Plus, all coursework will be handled on GitHub Classroom.

Requirements

- Ideally, CS350 Introduction to Software Engineering.
  - Lifecycle activities: requirements engineering, design, modelling, implementation, integration, testing, evolution
  - Experience of software design and programming (you will do some)
- Also, a computer, as we will do some live coding and hands-on activities.
  - If this is an issue, please contact me via email.

Assignments

- All five assignments will be handled by GitHub Classroom:
  - You will be given public test cases that you can freely execute
- All assignments are to be done individually.

Assignments

- Assignment 0: onboarding to GitHub Classroom - **due March 4th (no marks)**
- Assignment 1: Introduction to Metaprogramming - **due March 13th (5%)**
- Assignment 2: Coverage Profiler - **due April 10th (20%)**
- Assignment 3: Concolic Engine - **due May 8th (15%)**
- Assignment 4: Mutation Analysis - **due May 22th (10%)**
- Assignment 5: Hierarchical Delta Debugging - **due June 10th (10%)**
- **Assignment 0 is open now; 1~5 will be open later, asap.**

Assignments

- All assignments will come with grading test cases for GitHub Classroom: these will make up for 60% of the grade reserved to that assignment
- The remaining 40% is based on report and coding quality.
- Report quality: good writing and formatting, detailed description of what was done, etc
- Coding quality: good formatting, helpful commenting whenever appropriate, good design / architecture, etc

Projects and Teams

- Project: implement, and evaluate, an automated software testing technique and report the findings
- Submission and participation: I will receive a GitHub repository as a deliverable, and I expect to see everyone contributing to the project implementation.
  - Don’t say “we worked on a single machine, which is why all commits are from X”
  - Don’t say “I just worked on documentation and presentation”
- Until the number of remaining people makes it necessary, I intend to only accept teams of four people.

Projects and Teams

- We will start talking about teams once course registration is final - after those who want to drop actually drop :)
- Instead of mid-term, you will submit one page document of project ideas; after mid-term, you will read each other’s project ideas and form teams.
- Start thinking about what you want to implement now; feel free to discuss with me.

What is software testing?

Old Korean Saying

- “Try tapping the bridge before you cross, even if it is made of stone (돌다리도 두드려보고 건너라)”
- Roughly equivalent to “look before you leap” but… why tapping? :)

How do we know whether a software system is *correct*?

Rationalists vs. Empiricists

"It is correct because I proved that certain errors do not exist in the system"
(Formal Verification)

"It is correct because I tried it several times and it ran okay"
(Software Testing)

How do we know whether a software system is *correct*?

Rationalists vs. Empiricists

static

dynamic

Static Analysis

Overapproximation

- A *naive* static analysis will raise an alarm for division by zero here.
- Not being naive is expensive.

```
def foo(n):
    if n > 0:
        print(bar(n))
    else:
        return

def bar(a):
    return 42 / a
```

Dynamic Analysis

Underapproximation

- Testing inherently under-approximates program behaviour because we only make a partial observation (of the entire space of possible behaviour).

```
def bar(a):
    return 42 / a

def test_bar():
    assert bar(42) == 1
```

Why do we still keep doing it? :)

Assignment 0 is due next Monday!

References

- Strictly recommended for your reference: we *do not teach* these books and these books *do not contain answers* to this course.
  - Paul Ammann and Jeff Offutt. Introduction to Software Testing (2nd Ed.)
  - Andreas Zeller. Why Programs Fail (2nd Ed.)
  - Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. IEEE transactions on software engineering, 37(5):649–678.
  - P. McMinn. Search-based software test data generation: A survey. Software Testing, Verification and Reliability, 14(2):105–156, June 2004.