CSE P 590
Beyond Coverage: Modern Testing and Debugging
Spring 2019

Course introduction
April 02, 2019

The CSE P 590 team
Instructor
● René Just
● Office: CSE2 338
● Office hours: After class and by appointment
● rjust@cs.washington.edu

Teaching assistant
● Martin Kellogg
● Office: CSE2 253
● Office hours: Before/after class and by appointment
● kelloggm@cs.washington.edu

Today
● Logistics
● Brief introduction
● Your background and expectations
● Course overview
● Static vs. dynamic program analysis
● Class projects

Logistics
● CSE2 G10, Tue, 6:30pm – 9:20pm.
● Lectures, discussions, and lab session.
● Course material, schedule, etc. on website:
● Submission of assignments via Canvas:
  https://canvas.uw.edu
● Discussions on Piazza:
  piazza.com/washington/spring2019/csep590
My background

My research areas

- Software testing and verification
- Software debugging
- Software security

- Empirical software engineering
- Data science / Applied ML
My background

My research areas
- Software testing and verification
- Software debugging
- Software security
- Empirical software engineering
- Data science / Applied ML

The Role of Software Engineering in Research

Experimental infrastructure is software, too!

Example (automated debugging)
- 150 configurations, 1000+ benchmarks
- 1-85 hours per execution
- 200,000+ CPU hours (~23 CPU years)

Your background and expectations

Introduction and a very brief survey
- **Role**: What is your current role?
- **Background**: What is your SE background?
- **Top-2 expectations**: What do you expect from this course?

Today

- Logistics
- Brief introduction
- Your background and expectations
- Course overview
- Static vs. dynamic program analysis
- Class projects
Course overview: the big picture

- 04/02: Course introduction
- 04/09: Best practices and version control
- 04/16: Coverage-based testing
- 04/23: Automated test generation
- 04/30: Mutation-based testing
- 05/07: Mutation-based testing
- 05/14: Formal methods/constraint-based testing
- 05/21: Fault localization
- 05/28: Defect prediction
- 06/04: Type checking and pluggable types

Course overview: grading

- 30% Class project
- 60% In-class exercises (6 sessions)
- 10% Participation

Course overview: expectations

- Conduct a quarter-long group project.
- Some programming (and OO) experience.
- Read a few research papers.
- Have fun!
Today

- Logistics
- Brief introduction
- Your background and expectations
- Course overview
- Static vs. dynamic program analysis
- Class projects

What is Software Engineering?

- Developing in an IDE and software ecosystem?
- Coding and debugging?
- Deploying and running a software system?
- Empirical evaluations?
- Modeling and designing?

All of the above -- much more than just writing code!
What is Software Engineering?

**More than just writing code**
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

What is program analysis?

**More than just writing code**
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

Program analysis is a crucial task in Software Engineering!
What is program analysis?

- (Automatically) analyze the behavior of a program
  - optimize the program or
  - check program’s behavior (against its specification)
- Concerned with properties such as
  - Correctness
  - Safety
  - Liveness
  - Performance
- Can be static or dynamic or a combination of both

What’s the difference between a static analysis and a dynamic analysis?

Static vs. dynamic analysis

**Static analysis**
- Reason about a program without executing it
- Build an abstraction of run-time states (and prove a property of the program)

**Dynamic analysis**
- Reason about a program by executing it (with some inputs)
- Observe actual behavior

Why do we need program analysis?

- ~15 million lines of code
Let’s say 50 lines per page (0.05 mm)
Why do we need program analysis?

- ~15 million lines of code
- Let's say 50 lines per page (0.05 mm)
  - 300000 pages
  - 15 m (49 ft)
Why do we need program analysis?
- Increase confidence in program correctness
- Understand the program's behavior
- Prove properties about the program

Why do we need program analysis?
- Increase confidence in program correctness
- Understand the program's behavior
- Prove properties about the program

Code review/inspection
Different types of reviews
- Code/design review
- Informal walkthrough
- Formal inspection

Let's do an informal code review.
Anything that could be improved in this code?

Code review/inspection
Different types of reviews
- Code/design review
- Informal walkthrough
- Formal inspection

Let's do an informal code review.
Anything that could be improved in this code?

Code review/inspection
Different types of reviews
- Code/design review
- Informal walkthrough
- Formal inspection

Let's do an informal code review.
Anything that could be improved in this code?

Code review/inspection
Different types of reviews
- Code/design review
- Informal walkthrough
- Formal inspection

Let's do an informal code review.
Anything that could be improved in this code?

Double foo(double[] d) {
  int n = d.length;
  double s = 0;
  int i = 0;
  while (i<n)
    s = s + d[i];
    i = i + 1;
  double a = s / n;
  return a;
}

Double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;
  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;
  double avg = sum / n;
  return avg;
}
Code review/inspection

Different types of reviews

- Code/design review
- Informal walkthrough
- Formal inspection

Pros
- Can be applied at any step in the development process
- Improves confidence and communication

Cons
- Time-consuming
- Mostly informal
- Not replicable

static OSStatus
SSLVerifySignedServerKeyExchange(...) {
    OSStatus err;
    ...
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
    goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    err = sslRawVerify(ctx, ctx->peerPubKey, dataToSign, dataToSignLen, signature, signatureLen);
    if (err)
        sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify returned %d\n", (int)err);
        goto fail;
    }
    fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}

Anything wrong with that code?
Static vs. dynamic analysis

Static analysis
- Reason about a program without executing it
- Build an abstraction of run-time states (and prove a property of the program)

Dynamic analysis
- Reason about a program by executing it (with some inputs)
- Observe actual behavior

Static analysis: examples
- Type checking of a compiler
- Rule/pattern-based analysis (PMD, Findbugs, etc.)

```java
double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    int i = 0;
    while (i<n) {
        sum = sum + nums[i];
        i = i + 1;
    }
    double avg = sum / n;
    return avg;
}
```

Static analysis: examples
- Control-flow analysis
- Data-flow analysis

What is the control flow graph (CFG) for this avg function?
Static analysis: examples

- Control-flow analysis
- Data-flow analysis

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;
  int i = 0;
  while (i < n)
    sum = sum + nums[i];
  i = i + 1;
  double avg = sum / n;
  return avg;
}
```

Dynamic analysis: examples

- Software testing
- Software monitoring or profiling

```
A test for the avg function:

@Test
public void testAvg() {
  double nums = new double[]{1.0, 2.0, 3.0};
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected, actual, EPS);
}
```

Can we conclude that this is an infinite loop? Why or why not?

What happens if we execute this test? What can we conclude?
Static analysis vs. dynamic analysis

- Can these analyses pinpoint a problem in the code?
- Does a reported error always indicate that something is wrong with the code (no false positives)?
- Does no reported error indicate that there is nothing wrong with the code (no false negatives)?

Should we use static or dynamic analysis?

Class projects: overview

Logistics
- 3-5 students per project group.
- Group selection until 04/09 (further discussion on Piazza).
- 2 informal (in-class) presentations (~10min + Q&A).

High-level topics (suggestions)
1. Code coverage
   - A new code coverage tool for Java
   - API for existing code coverage tools (Cobertura, JaCoCo)
2. Mutation testing (Major)
   - Compiler-integrated mutator (compiler plugin)
   - Mutation analyzer (standalone or IDE plugin)
   - Visualization for mutation testing results
3. Fault database/benchmark (Defects4J)
   - Build system inference
   - Commit minimization
4. Static analysis: pluggable type checker
Project: New code coverage tool for Java

Goal:
Design and implement a new code coverage tool for Java programs (source-code, AST, or byte-code level).

Support queries such as:
- Is line x covered in method y?
- How often is it covered?
- How many lines are covered overall?
- How many lines exist in method y?
- ...

Project: API for existing code coverage tools

Goal:
Design a Java API that defines a common abstraction for code coverage tools, and support existing tools (e.g., Cobertura, JaCoCo).

Support queries such as:
- Is line x covered in method y?
- How often is it covered?
- How many lines are covered overall?
- How many lines exist in method y?
- ...

Projects: Mutation testing

Goal (project 1):
Develop a program mutator (e.g., Java compiler plugin).

Goal (project 2):
Develop a mutation analyzer (standalone, IDE plugin) for an existing program mutator.

Goal (project 3):
Develop a visualization for the output of an existing mutation analyzer.

Project: Build system inference

Goal:
Given a project’s build file (e.g., Apache Ant’s build.xml), automatically determine (infer) relevant properties.

```
<project name="Example" default="compile" basedir=".">

<!-- Compile the project -->
<target name="compile" depends="init" description="Compile">
<javac includeantruntime="true" srcdir="src" destdir="bin" debug="yes">
  <classpath location="lib/junit.jar"/>
</javac>
</target>
```

- Where are the sources?
- Where are the tests?
- What’s the classpath?
- ...
**Project: Commit minimization**

**Goal:**
Given a bug-fixing commit and a test suite that failed before and passes after that commit, automatically minimize the changes in that commit such that only changes relevant to the bug fix remain.

```
V_{n-1} \quad \text{features & refactorings} \quad V_{\text{bug}} \quad \text{isolated bug fix} \quad V_{\text{fix}} = V_n
```

---

**Class projects: overview**

**Logistics**
- 3-5 students per project group.
- Group selection until 04/09 (further discussion on Piazza).
- 2 informal (in-class) presentations (~10min + Q&A).

**High-level topics (suggestions)**

1. **Code coverage**
   a. A new code coverage tool for Java
   b. API for existing code coverage tools (Cobertura, JaCoCo)

2. **Mutation testing (Major)**
   a. Compiler-integrated mutator (compiler plugin)
   b. Mutation analyzer (standalone or IDE plugin)
   c. Visualization for mutation testing results

3. **Fault database/benchmark (Defects4J)**
   a. Build system inference
   b. Commit minimization

4. **Static analysis: pluggable type checker**

---

**Class projects: brainstorming session**

**Group by high-level interest**
- Code coverage
- Mutation testing
- Applied machine learning
- Static analysis
- Fault database/benchmarks

**Goals**
- What high-level project ideas should we add to the list?
- Pitch a brief project proposal for new ideas.