CSE 331
Software Design & Implementation

Topic: Software Tools

💬 Discussion: What’s a movie or show that you’ve enjoyed recently?
Reminders

• Before lecture today, we had office hours
• Today’s lecture is experimental

Upcoming Deadlines

• HW4 due Thursday (7/13)
Late Days

“No questions asked” late day policy:
- No more than one late day per assignment.
- No more than six late days total during the quarter.

“Questions asked” policy:
- Email us if you need more time
- Potential Downsides:
  • we may not be able to get you feedback quickly
  • you may fall behind on future assignments
Some quick reasoning...

Assertion 1: Students feel motivated to cheat in high-stress environments.
Assertion 2: Many of you find CSE 331 to be a high-stress environment.

=> Many of you feel motivated to cheat

**Don’t do it!**
- academically dishonest
- it won’t get you a high grade on an assignment
- it will build an unhealthy reliance and degrade your thinking

Instead come to talk to the course staff. We’ll help you!
Last Time...

• Design Principles
• Design in Java
• Style

Today’s Agenda

• Software Tools
• Tools for Testing
  • Test-case Ordering
  • Mutation Testing
• Other Tools
Software Tools
What is high quality?

Code is high quality when it is

1. **Correct**
   Everything else is of secondary importance

2. Easy to **change**
   Most work is making changes to existing systems

3. Easy to **understand**
   Needed for 1 & 2 above
How do we ensure correctness?

Best practice: use three techniques (we’ll study each)

1. **Tools**
   - type checkers, test runners, etc.

2. **Inspection**
   - think through your code carefully
   - have another person review your code

3. **Testing**
   - usually >50% of the work in building software

Together can catch >97% of bugs.
What is a software tool?

A **tool** is something that helps us write high-quality software.
- Forward/backward reasoning
- AFs, RIs, and ADTs

A **software tool** is a piece of software that helps us write high-quality software
- Describes a very large class of things
- We’ve seen a couple of these
- E.g. Git, IntelliJ, IntelliSense, Java compiler
What is a software tool?

FiniteSet.java

SimpleSet.java

...
What is a software tool?

How do people build software tools?

1. Identify a problem
2. Understand how developers currently solve it
3. Attempt to automate that process

In order to automate it, we need to define the solution precisely.

Until recently...
What is a software tool?

Disclaimer: I am not an expert!

If you find this work interesting, talk to the experts on campus
- UW PLSE, https://uwplse.org/
- UW NLP, https://www.cs.washington.edu/research/nlp
- Consider joining research https://www.cs.washington.edu/findingresearch
Tools for Testing
Testing so far…

In practice, to make a good test suite for a function we need

1. A way to make test cases
2. A way to determine if we have enough test cases

An algorithm to generate test suites:

```
suite = []
while (not enough test cases) {
    test = ...  // make a new test
    suite.add(test)
}
```
Brainstorm: Testing

How could we automate test case generation?

```
FiniteSet.java
SimpleSet.java
...  Software Tool  ...
Test 1
Test 2
Test 3
...  ...  ...  ...  ...
```
Test Generation: History

We can make test cases by reusing the input data from clients.

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
</table>

CSE 331 Summer 2023
Test Generation: Random

We can make test cases by randomly picking elements from our input space.

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
</table>

CSE 331 Summer 2023
Recall: Example

// returns: x < 0 => returns -x
// otherwise => returns x

int abs(int x) {
    if (x < -2) return -x;
    else return x;
}

What test cases might we want to consider for our test suite?

{..., -4, -3, -2, -1, 0, 1, 2, 3, ...}

is our entire input space.
## Test Generation: Random

We can make test cases by randomly picking elements from our input space.

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
</table>

Sometimes called fuzzing.
Test Generation: Random Objects

We can make test cases by randomly applying method calls to an object.
Test Generation: Specifications

We can make test cases by reading the specification.

<table>
<thead>
<tr>
<th>Benefits</th>
<th>Drawbacks</th>
</tr>
</thead>
</table>

Test-case Ordering

Does the order that we execute test cases matter?

We usually prefer to prioritize failing test cases.
- Investigate failures, not successes
- Failed test cases tend to fail early
**Code Coverage**

**Naive Attempt:** how many lines of code did we run?

```python
assert isEven(2)
assert isEven(4)
```

coverage = 3/5 = 60%

```python
function isEven(x):
    if (x % 2 == 0):
        return true
    else:
        return false
```
Code Coverage

**Naive Attempt**: how many lines of code did we run?

```python
assert isEven(2)
assert !isEven(3)

coverage = 100%
```

```python
function isEven(x):
    if (x % 2 == 0):
        return true
    else:
        return false
```
Code Coverage

**Naive Attempt:** how many lines of code did we run?

```python
function isEven(x):
    if (x % 2 == 0):
        return true
    else:
        return false
```

```python
isEven(2)
isEven(3)
```

coverage = 100%
(even though tests do nothing!)
Mutation Testing

**Better Attempt**: let’s introduce bugs into our code by making “mutant” programs

```plaintext
function isEven(x):
    if (x % 2 == 0):
        return true
    else:
        return false
```

Mutant #1

```plaintext
function isEven(x):
    if (x % 2 == 1):
        return true
    else:
        return false
```

Mutant #2

```plaintext
function isEven(x):
    if (x % 2 != 0):
        return true
    else:
        return false
```

Note: Need to define allowed mutations
**Mutation Testing**

**Better Attempt:** let’s introduce single-line bugs into our code (i.e. mutants)

```python
assert isEven(2)
assert isEven(3)

mutants score = 100%
```

Mutant #1:

```python
function isEven(x):
    if (x % 2 != 0):
        return true
    else:
        return false
```

Mutant #2:

```python
function isEven(x):
    if (x % 2 == 1):
        return true
    else:
        return false
```
Mutation Testing so far...

In practice, to make a good test suite for a function we need

1. A way to make test cases
2. A way to determine if we have enough test cases

[mutation score]

An algorithm to generate test suites:

```
suite = []
while (undetected mutants) {
    mutant = ...  // introduce a bug that breaks our tests
    test = ...    // make a test that catches that bug
    suite.add(test)
}
```
A subdomain is *revealing* for error $E$ if either:
- *every* input in that subdomain triggers error $E$, or
- *no* input in that subdomain triggers error $E$

Each test case produced with mutation testing reveals some bug!

So why don’t people use it in practice?
- Need to define the single-line mutations allowed
Other Tools

Correctness:
- Fault localization
- Program verification
- Program analysis
  - Static vs. dynamic
- Program synthesis

Changeability:
- Code generation

Understandability:
- Linters
Tools for Testing
Other Tools: Fault Localization

Given your software and a failing test identify where the bug is likely to be.

- Could be approximate (e.g. this region)
- Could be multiple answers
Other Tools: Automated Program Repair

Given your software and a failing test suite, identify a patch that fixes the code.
Other Tools: Program Verification

Given your software and formal specification, prove that code is correct.

- Model checking
- Deductive verification
Other Tools: Program Analysis

Given your software, identify if it has some property.
- Static analysis
  - Data-flow analysis for taint checking
- Dynamic analysis
  - Program slicing
Other Tools: Program Synthesis

Given a formal specification, identify a program that satisfies that implementation.
Other Tools

Changeability:
- Code generation
- Feedback

Understandability:
- Linters

Note: this list is actually very long!
Before next class...

1. Ask us questions about HW4!
   - Lots of good discussion on Ed

2. Section tomorrow will focus on HW5 preparation.