Exam 1 and HW 3 Review

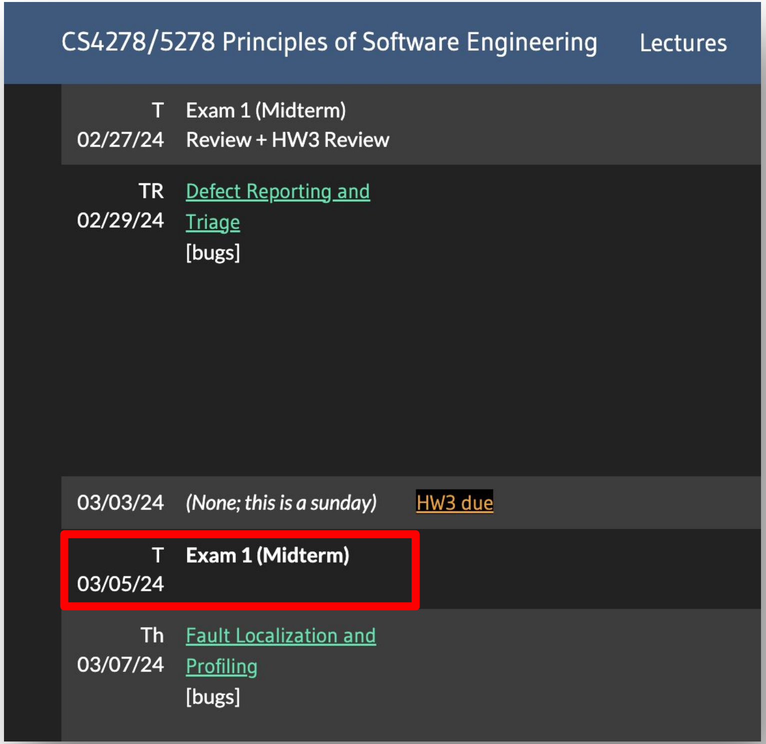
CS 4278/5278: Principles of Software Engineering

Eric Li
Undergraduate Teaching Assistant
jiliang.li@vanderbilt.edu

VANDERBILT
School *of* Engineering

Exam 1

- Tuesday March 5
- Class Time (75 min)
  - 1:15 PM - 2:30 PM
  - FGH 134
- TA-Proctored



VANDERBILT School of Engineering

Exam Structure

- Paper-based, written exam (9-12 pages)
- Bring a pen/pencil
- 100 points in total
- 6-7 multipart questions + 1 multipart bonus (4-6 pts)
- Short answer, answer bank, fill in the blank

Exam Structure

- Open-book, open-notes, open-internet

- NO ChatGPT
- NO collaborations/communications (e.g. online chatting)

General Tips

- The exam will be fast-paced
- So study in advance (e.g. there's no time to review/learn a concept on the spot)

Questions?

Exam Topics

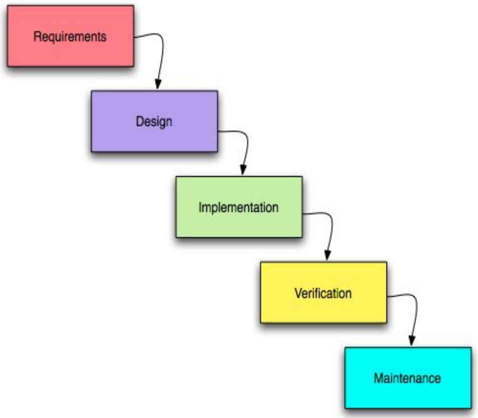
- Process, Risk, Scheduling
- Measurement and Quality Assurance
- Testing and Code Review
- Dynamic, Static, and Dataflow Analysis
- Defect Reporting and Triage

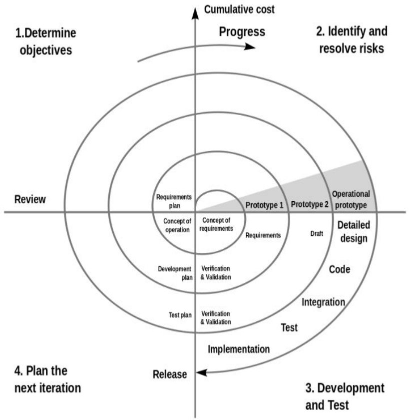
Exam Topics

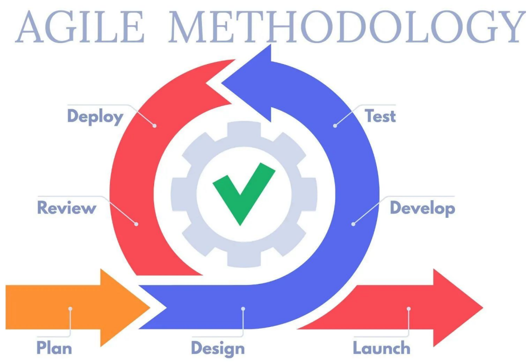
- Process, Risk, Scheduling
- Measurement and Quality Assurance
- Testing and Code Review
- Dynamic, Static, and Dataflow Analysis
- Defect Reporting and Triage

Process, Risk, Scheduling

- Waterfall, Spiral, Agile - advantages and shortcomings

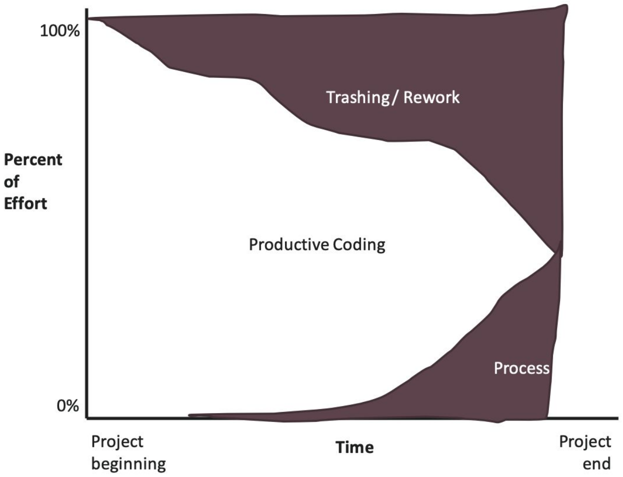






Process, Risk, Scheduling

- Defect cost vs. creation/detection time
- Result of failing to plan



Process, **Risk**, Scheduling

- Risks are everywhere in a software process
  - Staff illness or turnover, market competition, slow progress…

- **Risks**, along with a lack of data, leads to **uncertainties**
- Uncertainties can be reduced by **measurement**
  - More on this later…

Process, **Risk**, Scheduling

- Zero-risk bias
  - Prefer eliminating risk over larger reduction in risk
- Risk management is key to project management

Process, Risk, **Scheduling**

- Scheduling manages risk during project execution

- Scheduling is key to a software process
  - A project should plan time, cost, resources, etc.

Process, Risk, **Scheduling**

- Strategies to estimate time for a project
  - A constructive cost model (cocomo)
- Milestones vs. deliverables
  - Endpoint of a task vs. results for the customer
- “Almost done” problem

Process, Risk, Scheduling

- In general, know your good practices and bad practices!

Questions?

What’s the narratives so far?

➔ Software processes come with risks, which leads to uncertainty

➔ Measurement and quality assurance can reduce uncertainty

Exam Topics

- Process, Risk, Scheduling
- Measurement and Quality Assurance
- Testing and Code Review
- Dynamic, Static, and Dataflow Analysis
- Defect Reporting and Triage

Measurement and Quality Assurance

- Measurements measure all kinds of things
  - Software quality, process quality, funding, etc.
- Measurements assist decision making
  - Example: where should funds/effort be allocated?

Measurement and Quality Assurance

- Measurement of code quality
  - Maintainability Index
    - Halstead Volume
    - Cyclomatic Complexity
    - Lines of Code

Measurement and Quality Assurance

- Types of validity for a given metric
  - Construct, predictive, external
- Metric-based incentives
  - What could be a drawback?

Measurement and Quality Assurance

- McNamara Fallacy
  - Making decisions based solely on quantitative metrics
- Streetlight effect
  - Searching for something and looking only where it is easiest
- Statistics
  - False positive paradox, correlation != causation, confounding variables

Measurement and Quality Assurance

- Measurements / software metrics should be used carefully!

Measurement and Quality Assurance

- Example: you are working on a web development project.
  - What could be a **risk**?
  - What's the **uncertainty** associated with the risk?
  - How can a **measurement** be used to reduce that uncertainty?
  - In general, what are some good vs. bad practices?

Measurement and **Quality Assurance**

- Halting Problem in QA
  - We can never be sure a program is correct
- Testing can give us an estimate
  - Demonstrates the presence of bugs, not their absence

Questions?

Exam Topics

- Process, Risk, Scheduling
- Measurement and Quality Assurance
- Testing and Code Review
- Dynamic, Static, and Dataflow Analysis
- Defect Reporting and Triage

Testing and Code Review

- Types of testing
  - Regression - running old tests
  - Unit - test individual pieces
  - Integration - end-to-end testing
  - Fuzz - testing lots of random inputs
  - Penetration - testing for security vulnerabilities
  - Mocking - test with simulated (not real) objects

Testing and Code Review

- Coverage as a metric for test suite comparison
  - Branch, line, & path coverage
  - You should be able to calculate branch and line coverage
  - Is it easy to enumerate paths?
- Coverage instrumentation and relation to observer effect
  - Instrumenting a program could change its behavior

Testing and Code Review

- Mutation testing
  - Defect seeding to test quality of a test suite
  - Intentionally adding bugs, and then kill that mutant
- Mutation operator and mutant orders
- Competent programmer hypothesis & coupling effect
  - How do they relate to mutation testing?
- Equivalent mutants
- Know how to calculate mutation score

Testing and Code Review

- A test case consists of
  - An input (data), an oracle (expected output), and a comparator
- Test Input Generation (automatically)
  - Guided by line/branch/path enumeration
- Test Oracle Generation (automatically)
  - Oracle inference via invariants

Testing and Code Review

- Invariants and oracle inference
  - Invariants are predicate over expressions that is true on all executions
  - High quality/confidence invariants can serves as oracles
- Common vs correct behavior for invariant inference

Testing and Code Review

- Alpha Testing - by developers
- Beta Testing - by external users
- A/B testing - show impact of a difference in one feature

- Test suite minimization
  - How hard is it?

Testing and Code Review

- Unit Testing
  - You should know how to handwrite a unit test case with JUnit
  - Criteria for good unit test
  - @RepeatedTest, @Timeout, @BeforeEach, @ParameterizedTest
  - try / catch, assert
  - Think about boundary / empty / error cases

Testing and **Code Review**

- Inspection incentive and root cause analysis
  - Why inspect? To prevent problems from reoccurring
- Metrics on inspection
  - Accuracy, speed, focus fatigue, etc.
- Different types of code review
  - Formal inspection, walkthrough, **pair programming**, passaround, ad hoc

Testing and **Code Review**

- Code review
  - A second pair of eyes; find defects, improve quality
- Formal code inspection
  - A team effort; more formal and holistic
- Pull request
  - Proposed changes to merge into a repository

Questions?

Exam Topics

- Process, Risk, Scheduling
- Measurement and Quality Assurance
- Testing and Code Review
- Dynamic, Static, and Dataflow Analysis
- Defect Reporting and Triage

Dynamic, Static, and Dataflow Analysis

- Dynamic analysis - analyzing a program by running it
- Steps
  - Instrument the program at compile time (on source / binary code)
  - Run program systematically (controlled input or environment)
  - Monitor internal state at runtime
  - Analyze results (path coverage, information flow, profiling)

Dynamic, Static, and Dataflow Analysis

- Race condition
  - Output depends on sequence or timing of “uncontrollable” events
- Taint tracking using sources and sinks
- Dynamic analysis is very input dependent
- Dynamic analysis focuses on one property of output information

Dynamic, Static, and Dataflow Analysis

- Examples of dynamic analysis
  - Eraser - shared variable must be guarded by a lock
  - Chaos Monkey - random destructions of services
  - CHESS - tracks different combinations of thread interleaving
  - Driver Verifier - replaces default OS subroutines with others
- You need to understand what these tools are doing and what could possibly go wrong with each tool

Dynamic, Static, and Dataflow Analysis

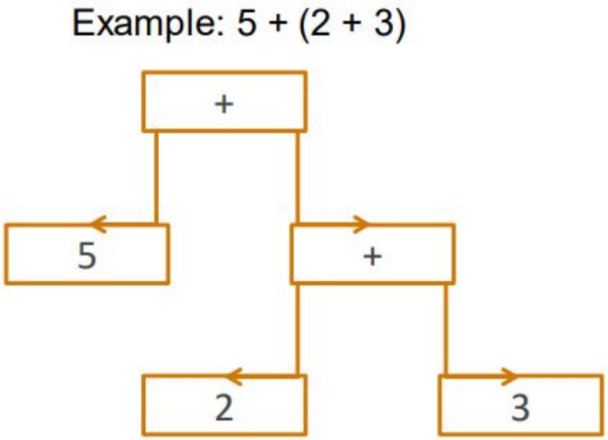
- Soundness vs. completeness
  - Sound analysis: no false negatives
    - e.g. all bugs are identified
  - Complete analysis: no false positives
    - e.g. all reported bugs are actually bugs

Dynamic, **Static, and Dataflow Analysis**

- Static analysis - analysis of code *not* at runtime
- Dataflow analysis - a popular approach to static analysis
- Main ideas
  - Abstraction as hiding unnecessary details to simplify program
  - Programs being simplified down to trees, graphs, or strings

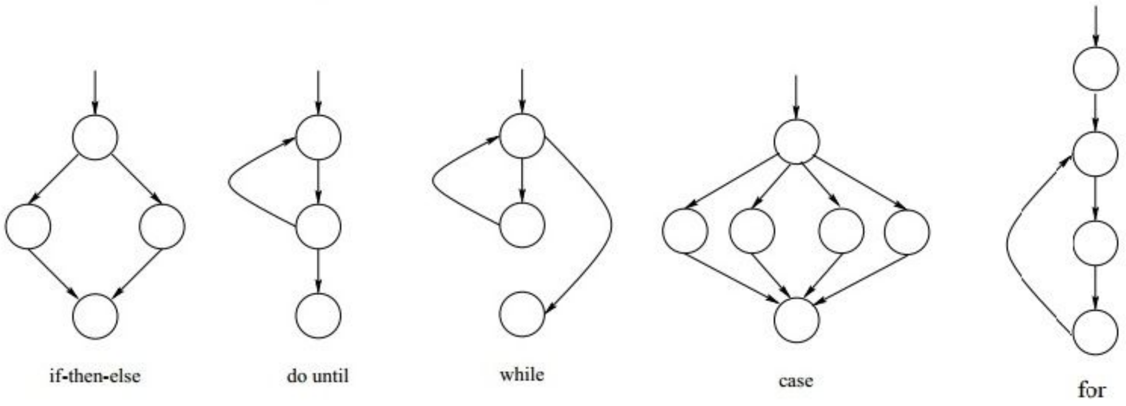
Dynamic, **Static, and Dataflow Analysis**

- Abstract Syntax Tree (AST) represents syntactic structure of source code; it records only semantically relevant information



Dynamic, **Static, and Dataflow Analysis**

- A control flow graph (CFG) is a graph representation of all paths that might be traversed through a program during execution

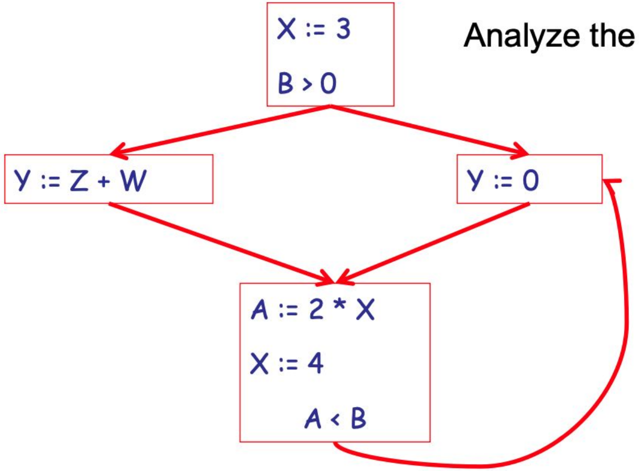


Dynamic, **Static, and Dataflow Analysis**

- Dataflow analysis
  - Gather information on the possible set of values at various points
  - Forward analysis
    - e.g. definitely null, constant propagation
  - Backward analysis
    - e.g. secure information flow, liveness

Dynamic, **Static, and Dataflow Analysis**

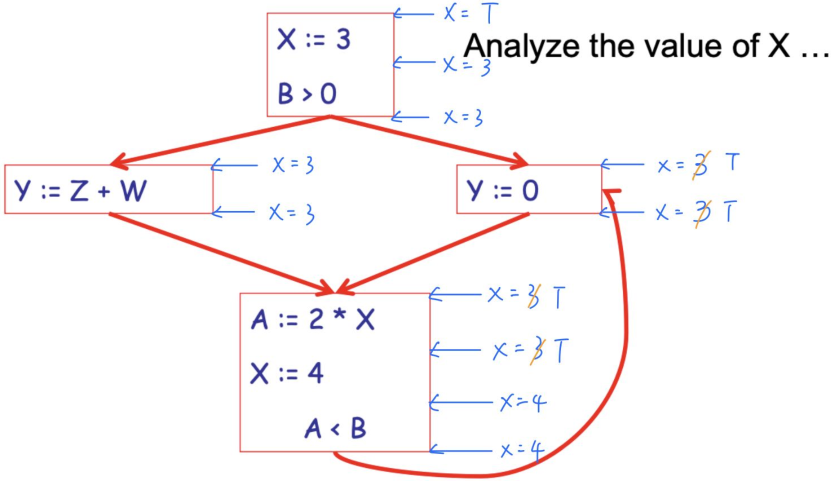
- Forward analysis



Analyze the value of X …

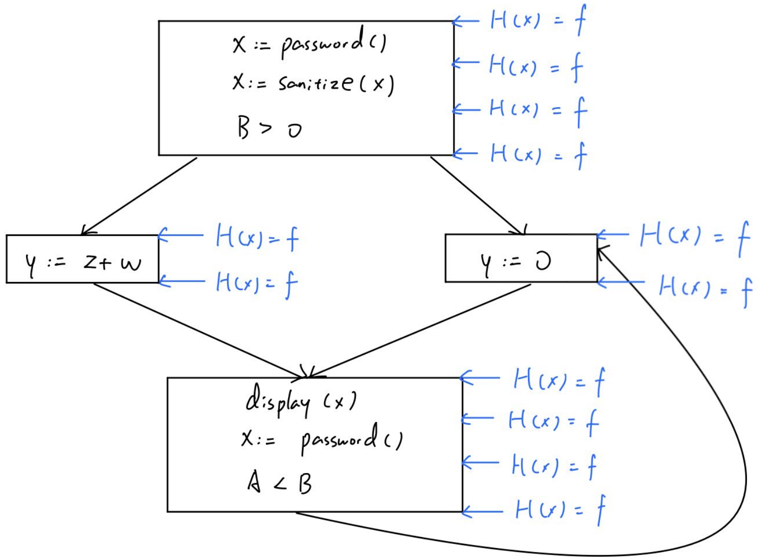
Dynamic, **Static, and Dataflow Analysis**

- Forward analysis



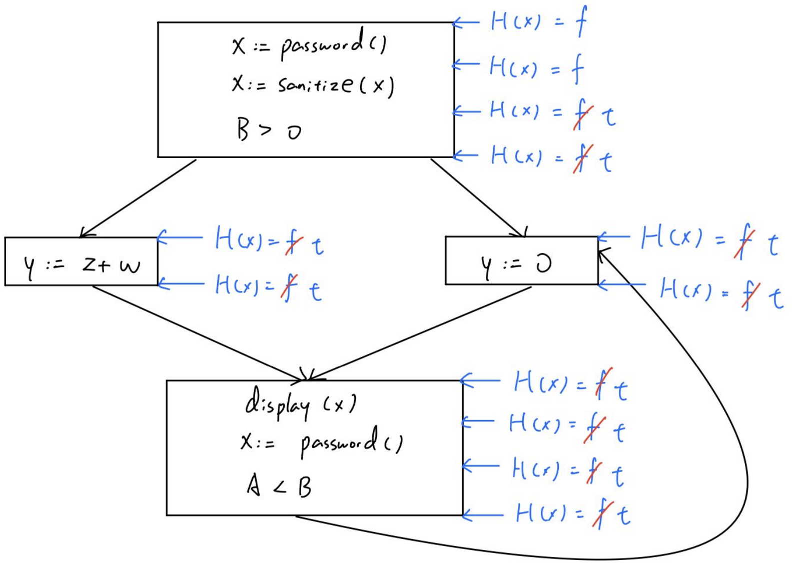
Dynamic, **Static, and Dataflow Analysis**

- Backward analysis



Dynamic, **Static, and Dataflow Analysis**

- Backward analysis



Dynamic, **Static, and Dataflow Analysis**

- Rice's Theorem and undecidability of program's properties
  - All of the interesting properties of a program are undecidable
  - Dataflow analysis is conservative program analyses (imprecision; okay to say we don't know)
- Does dataflow analysis always terminate?

Questions?

Exam Topics

- Process, Risk, Scheduling
- Measurement and Quality Assurance
- Testing and Code Review
- Dynamic, Static, and Dataflow Analysis
- Defect Reporting and Triage

Defect Reporting and Triage

- Will be covered next class

Miscellaneous

- Know your guest lecture materials
- Know your homework assignments
- May be beneficial to know optional readings (extra-credit)
- May be beneficial to know the trivia (extra-credit)

Defect Reporting and Triage

- Fault - exceptional situation at run time
- Defect - characteristic of a product which hinders its usability for its intended purpose
- Bug report - Accurately and precisely describe the bug and how to reproduce it
- Triage - measure of urgency

Questions?

Homework 3 Intro

CS 4278/5278: Principles of Software Engineering

Starting Point

If you haven’t started, please start as soon as possible. This one can take pretty long.

Starting Point

- Grading server uses **Python 3.5.2**
  - So newer features like f-strings are not supported
- Read documentation on the **ast** module and the **astor** module.
- You should submit a single file, “mutate.py”
  - The program should generate mutants that are named “0.py”, “1.py”, … (up to 100 files)
  - Other outputs are ignored

- Can someone quickly explain what mutation testing is?
  - hint: make sure to review mutation testing for the exam!

Mutation Operators

You should implement and support the following three mutation activities:

1. Negate any single comparison operators (>= becomes <, = becomes !=)
2. Swap binary operators +, and -, as well as * and //.
3. Delete an assignment or function call statement.

Held-Out Test Suites

- Test Suites A, B, C, D, and E have 92%, 91%, 90%, 88%, and 79% statement coverage of fuzzywuzzy, respectively. These suites have 80, 57, 47, 32, and 9 tests, respectively.
- **Swap Binary Operators** to distinguish Test Suite A and B from C, D, and E
- **Swap Comparison Operators** to distinguish between Suite B, C, D, and E
- **Delete Assignments and Function Calls** to distinguish C, D, and E. With care, to distinguish between A and B.
- **Higher-Order Mutation** may distinguish Test Suites B, C, and D.
- **Use Creativity** to distinguish between Test Suite A and B
  - hint: try changing assignments!

Starter Code

Look up the "HW3 example code" posted on Piazza

```
import ast
import astor
with open("xxx.py", "r") as src:
    # convert BinOp "+" to "-"
    tree = ast.parse(src.read())
    new_tree = AddTransformer().visit(tree0)  // how to write a transformer?
    file = astor.to_source(new_tree).strip()
    # then write to an output file
```

Starter Code

- How to write a Transformer?
  - Read the ast.NodeVisitor and ast.NodeTransformer sections in ast documentation
    - NodeTransformer is a subclass of NodeVisitor (recall ISD concepts…)
    - Use inheritance to create different transformers:
      - e.x., **class AddTransformer(ast.NodeTransformer)**
      - Transformer subclasses should have a visitor function (see documentation)
    - https://docs.python.org/3/library/ast.html
  - How do I parse a python file into an AST? How do I turn an AST into a source file?
    - Read documentation on ast.parse, ast.dump, astor.to_source, etc.
    - https://astor.readthedocs.io/en/latest/
- Is this the only approach?
  - No, previous students have tried several other approaches that worked well!
  - The transformer approach above is one that should be straightforward

Most Common Pitfall

- Don’t start with high order mutants and then adjust.
  - Most students who finished this assignment quickly started with low order mutants. Then, adjust your strategies and built up higher order mutants slowly based on your low order mutants.

- If you used ChatGPT, it tends to generate really high-order mutants from the start.

Common Pitfalls and Advices

1. Don't start with higher order mutants!
    a. Though carefully designed higher order mutants can be important, most higher order mutants have a high chance of being detected by every test suite.
2. Increasing the odds of one mutation operator also effectively reduces the odds of the others (since you can only produce a fixed number of mutants)
3. Be careful not to mistakenly share the tree data structure between mutants, as you may end up with more edits than you thought
4. Try making a high-quality test suite locally and evaluating against it.
5. Make sure you actually have a chance of mutating every relevant node.
6. You may want to implement additional mutation operators.
    a. See https://huang.isis.vanderbilt.edu/cs4278/readings/mutation-testing.pdf
7. After creating your mutants, you should run pylint to minimize the number of linting/syntax errors reported

VANDERBILT
School of Engineering

A Strategy that Worked with Many Students

- Key Insight:

The Transformer class inherits from the Visitor class, so they traverse the AST in the same order.

A Strategy that Worked with Many Students

- Don’t jump so fast into changing the AST. Instead, visit the tree first and check the operators of interest.
- Write a list containing the locations of all nodes of interest. The order is determined by the traversal order of the visitor.

A Strategy that Worked with Many Students

- Now, you have a list of the locations of all nodes of interest.
- Build your strategies using this list.
    - Keep track of what mutations worked and what doesn't

Low vs. High Order Mutants

- You can get full credit with
  - Only order 1 mutants
  - Only high order mutants
  - A combination of low and high order mutants

Interested in AST?

Take Compilers (CS 3276/5276)!