Test Suite Quality

CS 485/698: AI-Assisted SE

Today's Agenda

- Team meeting (~5 minutes)
  - Standup: how is P5 going? What else do you plan to build before the end of the semester (i.e., for P7)?
  - If your project team has two or more people, split yourselves into **offensive** and **defensive** sub-teams of approximately equal size.
    - I will stop by during the team meeting to ask
- Lecture on test suite quality metrics
- In-class activity: testing your tests

Test suite quality

Definition: a ***test suite*** is a collection of tests for the same program

Test suite quality

Definition: a ***test suite*** is a collection of tests for the same program

- What makes one test suite **better or worse** than another?
  - not just the sum of the “goodness” of all the individual tests!

Test suite quality

Definition: a ***test suite*** is a collection of tests for the same program

- What makes one test suite **better or worse** than another?
  - not just the sum of the "goodness" of all the individual tests!
- Why might we want to **automatically measure** the quality of a test suite?

Test suite quality

Definition: a ***test suite*** is a collection of tests for the same program

- What makes one test suite **better or worse** than another?
  - not just the sum of the "goodness" of all the individual tests!
- Why might we want to **automatically measure** the quality of a test suite?
  - Traditional answer: to show that we've done a good job testing the program

Test suite quality

Definition: a ***test suite*** is a collection of tests for the same program

- What makes one test suite **better or worse** than another?
  - not just the sum of the "goodness" of all the individual tests!
- Why might we want to **automatically measure** the quality of a test suite?
  - Traditional answer: to show that we've done a good job testing the program
  - 2026 answer: to give our AI agents a ***fitness function*** for the test suites that we're asking them to write

Aside: genetic algorithms and fitness functions

- ***Genetic algorithms*** are a class of biology-inspired algorithms that “evolve” a solution to a problem

Aside: genetic algorithms and fitness functions

- ***Genetic algorithms*** are a class of biology-inspired algorithms that "evolve" a solution to a problem
  - Maintain a ***population*** of possible solutions

Aside: genetic algorithms and fitness functions

- ***Genetic algorithms*** are a class of biology-inspired algorithms that "evolve" a solution to a problem
  - Maintain a ***population*** of possible solutions
  - ***Mutation operators*** combine (parts of) solutions from the population to create a new generation of solutions

Aside: genetic algorithms and fitness functions

- ***Genetic algorithms*** are a class of biology-inspired algorithms that "evolve" a solution to a problem
  - Maintain a ***population*** of possible solutions
  - ***Mutation operators*** combine (parts of) solutions from the population to create a new generation of solutions
  - A ***fitness function*** prunes the starting population + the new generation back down

Aside: genetic algorithms and fitness functions

- ***Genetic algorithms*** are a class of biology-inspired algorithms that "evolve" a solution to a problem
  - Maintain a ***population*** of possible solutions
  - ***Mutation operators*** combine (parts of) solutions from the population to create a new generation of solutions
  - A ***fitness function*** prunes the starting population + the new generation back down
  - Repeat until some stopping condition

Aside: genetic algorithms and fitness functions

- What makes a **good** fitness function?

Aside: genetic algorithms and fitness functions

- What makes a **good** fitness function?
  - Continuous
  - Monotonic (or at least with few local optima)
  - Cheap to evaluate

Aside: genetic algorithms and fitness functions

- What makes a **good** fitness function?
  - Continuous
  - Monotonic (or at least with few local optima)
  - Cheap to evaluate
- Why might we want a fitness function for an AI agent?

Aside: genetic algorithms and fitness functions

- What makes a **good** fitness function?
  - Continuous
  - Monotonic (or at least with few local optima)
  - Cheap to evaluate
- Why might we want a fitness function for an AI agent?
  - We can “put the agent in a loop” with a concrete goal and then let it run autonomously
  - Gives us a way to evaluate the quality of the agent’s work without necessarily understanding the details

Test Suite Quality Metrics

- How can we automatically measure test suite quality?
- Today we will cover two approaches:
  - **Coverage**: how much of the program's behavior have we actually checked?
  - **Adversarial**: if we intentionally seed bugs into the program, does the test suite actually detect them?

Coverage: statement coverage

Definition: ***Statement coverage*** is the fraction of source statements that are executed by the test suite.

Coverage: statement coverage

Definition: ***Statement coverage*** is the fraction of source statements that are executed by the test suite.

- **Key Logical Observation**: If we **never test** line X then testing **cannot rule out** the presence of a bug on line X

Coverage: statement coverage

Definition: ***Statement coverage*** is the fraction of source statements that are executed by the test suite.

- **Key Logical Observation**: If we **never test** line X then testing **cannot rule out** the presence of a bug on line X
- Example: if our test executes lines 1 and 2, but there is a bug on line 3, there is **no way** that our test will find the bug!

Coverage: statement coverage

Definition: ***Statement coverage*** is the fraction of source statements that are executed by the test suite.

- **Key Logical Observation**: If we **never test** line X then testing **cannot rule out** the presence of a bug on line X
- Example: if our test executes lines 1 and 2, but there is a bug on line 3, there is **no way** that our test will find the bug!
- Most languages have one or more test frameworks that provide statement coverage instrumentation
    - e.g., Jest for JavaScript, gcov for C, Cobertura for Java

Coverage: statement coverage

- Statement coverage is **fundamentally limited**:

Coverage: statement coverage

- Statement coverage is **fundamentally limited**:
  - Only need to visit each statement once to get credit for it
    - What if a statement can behave differently depending on context?

Coverage: statement coverage

- Statement coverage is **fundamentally limited**:
  - Only need to visit each statement once to get credit for it
    - What if a statement can behave differently depending on context?
  - Statement coverage doesn't consider whether the program got the right answer
    - Actually, it doesn't evaluate the oracle **at all**
      - Really, it measures how good your ***inputs*** are, not how good your tests are!

Coverage: statement coverage

- Statement coverage is **fundamentally limited**:
  - Only need to visit each statement once to get credit for it
    - What if a statement can behave differently depending on context?
  - Statement coverage doesn't consider whether the program got the
    - Act
      - how good your tests are!

We can address the first problem with better coverage metrics, but for the second problem we need a completely different approach

Better coverage metrics: branch coverage

Definition: ***Branch coverage*** is a test suite quality metric that counts the total number of conditional branches exercised by that test suite (i.e., if true and if false are counted separately)

Better coverage metrics: branch coverage

Definition: ***Branch coverage*** is a test suite quality metric that counts the total number of conditional branches exercised by that test suite (i.e., if true and if false are counted separately)

- Branch coverage gives **more confidence** than statement coverage
  - In fact, 100% branch coverage **implies** 100% line coverage
    - Why?

Better coverage metrics: branch coverage

Definition: ***Branch coverage*** is a test suite quality metric that counts the total number of conditional branches exercised by that test suite (i.e., if true and if false are counted separately)

- Branch coverage gives **more confidence** than statement coverage
  - In fact, 100% branch coverage **implies** 100% line coverage
    - Why?
- However, branch coverage is "**more expensive**" in the sense that it is harder for a test suite to have high branch coverage than to have high statement coverage

Beyond branch coverage

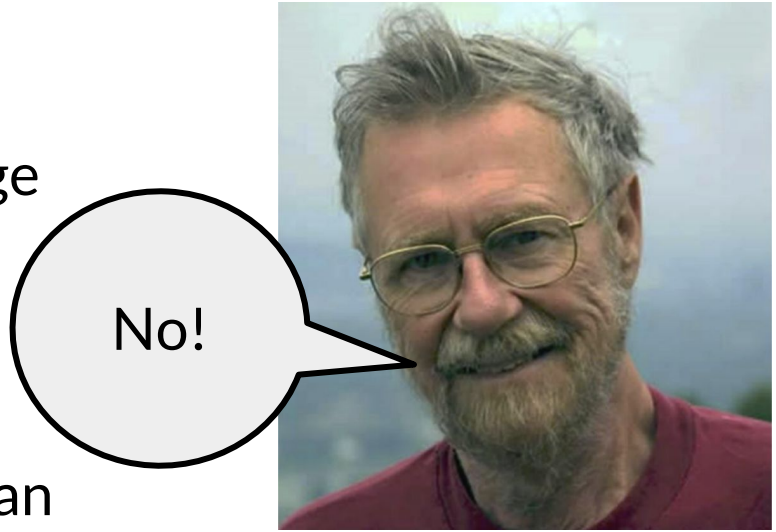
- We know 100% statement coverage **doesn’t guarantee** no bugs

Beyond branch coverage

- We know 100% statement coverage **doesn't guarantee** no bugs
  - what about **100% branch coverage**? If we have 100% branch coverage, does that mean no bugs?

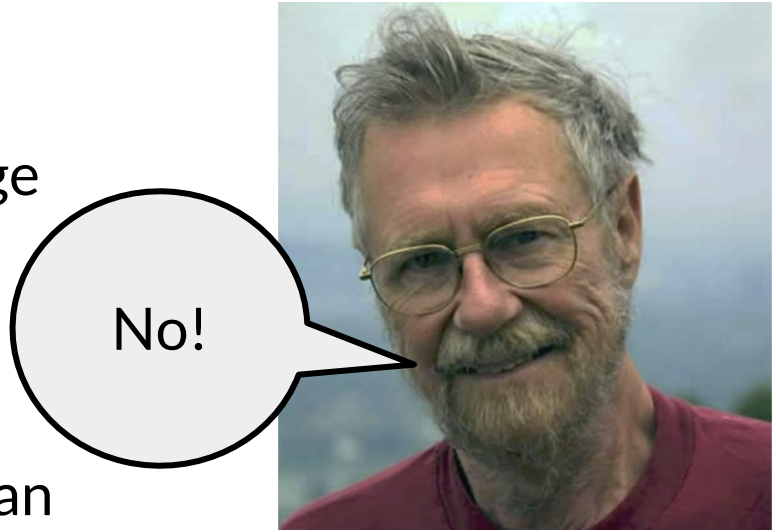
Beyond branch coverage

- We know 100% statement coverage **doesn't guarantee** no bugs
  - what about **100% branch coverage**? If we have 100% branch coverage, does that mean no bugs?



Beyond branch coverage

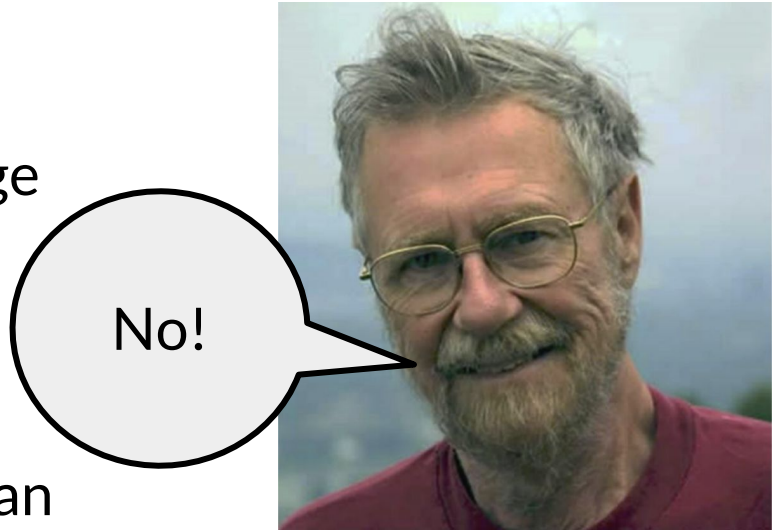
- We know 100% statement coverage **doesn't guarantee** no bugs
  - what about **100% branch coverage**? If we have 100% branch coverage, does that mean no bugs?



- Recall: "tests can only show the presence of bugs, not their absence"

Beyond branch coverage

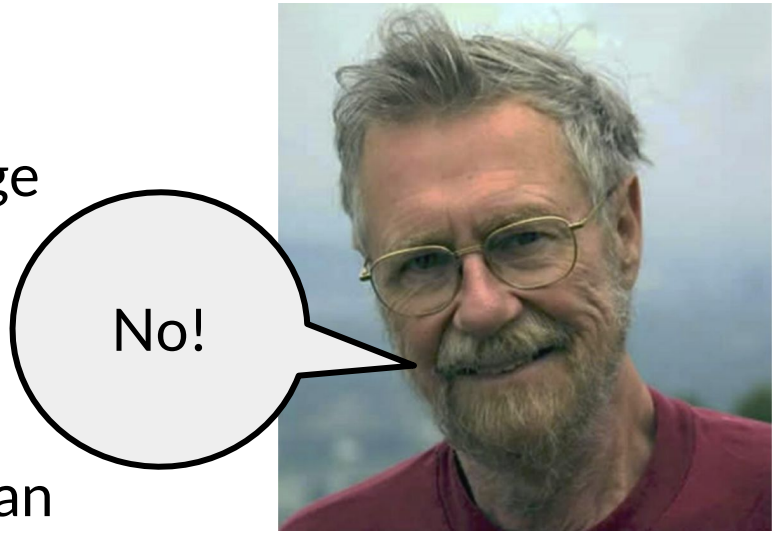
- We know 100% statement coverage **doesn't guarantee** no bugs
  - what about **100% branch coverage**? If we have 100% branch coverage, does that mean no bugs?



- Recall: "tests can only show the presence of bugs, not their absence"
- More coverage = more confidence, but **no guarantees**!

Beyond branch coverage

- We know 100% statement coverage **doesn't guarantee** no bugs
  - what about **100% branch coverage**? If we have 100% branch coverage, does that mean no bugs?



- Recall: "tests can only show the presence of bugs, not their absence"
- More coverage = more confidence, but **no guarantees**!
- Can we get **finer-grained** than branch coverage?

Coverage: other kinds of coverage

- **Function Coverage**: what fraction of functions have been called?

Coverage: other kinds of coverage

- **Function Coverage**: what fraction of functions have been called?
- **Condition Coverage**: what fraction of boolean subexpressions have been evaluated to both true and also (e.g., on another run) to false?
    - Comparing this to branch coverage is a not-uncommon test question …

Coverage: other kinds of coverage

- **Function Coverage**: what fraction of functions have been called?
- **Condition Coverage**: what fraction of boolean subexpressions have been evaluated to both true and also (e.g., on another run) to false?
    - Comparing this to branch coverage is a not-uncommon test question …
- **Modified Condition / Decision Coverage** (MC/DC): condition coverage plus more
    - Used in mission critical (e.g., avionics) software

Adversarial Test Suite Quality Metrics

- *“**Quis custodes ipsos custodiet?**”*
  - Decimus Ivnivs Ivvenalis (“Juvenal”), Roman satirist

Adversarial Test Suite Quality Metrics

- *“**Quis custodes ipsos custodiet?**”*
  - Decimus Ivnivs Ivvenalis (“Juvenal”), Roman satirist
- Usually translated into English as “***who watches the watchers?***”

Adversarial Test Suite Quality Metrics

- *“**Quis custodes ipsos custodiet?**”*
  - Decimus Ivnivs Ivvenalis (“Juvenal”), Roman satirist
- Usually translated into English as “***who watches the watchers?***”
- There is a **general technique** for solving “who watches the watchers”-style problems: intentionally introduce a small number of known-in-advance problems into the system…

Adversarial Test Suite Quality Metrics

- *“**Quis custodes ipsos custodiet?**”*
  - Decimus Ivnivs Ivvenalis (“Juvenal”), Roman satirist
- Usually translated into English as “***who watches the watchers?***”
- There is a **general technique** for solving “who watches the watchers”-style problems: intentionally introduce a small number of known-in-advance problems into the system…
  - …and then see whether the “watchers” **actually detect** the known problems!

Adversarial Test Suite Quality Metrics

- *"**Quis custodes ipsos custodiet?**"*
  - Decimus Ivnivs Ivvenalis ("Juvenal"), Roman satirist
- Usually translated into English as "***who watches the watchers?***"
- There is a **general technique** for solving "who watches the watchers"-style problems: intentionally introduce a small number of known-in-advance problems into the system…
  - …and then see whether the "watchers" **actually detect** the known problems!
- In software testing, the tests are the watchers. The application of this general technique to testing is called ***mutation testing***

Mutation testing

Definition: ***Mutation testing*** (or ***mutation analysis***) is a test suite adequacy metric in which the quality of a test suite is related to the number of intentionally-added defects it finds

Mutation testing

Definition: ***Mutation testing*** (or ***mutation analysis***) is a test suite adequacy metric in which the quality of a test suite is related to the number of intentionally-added defects it finds

- Informally: “You claim your test suite is really great at finding security bugs? Well, I'll just **intentionally add a security bug** to my source code and see if your test suite finds it!”
- Why is this better than coverage?

Mutation testing

Definition: ***Mutation testing*** (or ***mutation analysis***) is a test suite adequacy metric in which the quality of a test suite is related to the number of intentionally-added defects it finds

- Informally: “You claim your test suite is really great at finding security bugs? Well, I'll just **intentionally add a security bug** to my source code and see if your test suite finds it!”
- Why is this better than coverage?
  - Evaluates both coverage of inputs and **quality of oracles** implicitly: to find the seeded bug you must both reach the line and detect that something is wrong!

Mutation testing: how it works

- A ***mutant*** is a copy of a program to which we have intentionally added a defect

Mutation testing: how it works

- A ***mutant*** is a copy of a program to which we have intentionally added a defect
- A test suite is said to ***kill*** (or ***detect***, or ***reveal***) a mutant if the mutant fails a test that the original passes.

Mutation testing: how it works

- A ***mutant*** is a copy of a program to which we have intentionally added a defect
- A test suite is said to ***kill*** (or ***detect***, or ***reveal***) a mutant if the mutant fails a test that the original passes.
- Mutation testing of a test suite proceeds by making a number of mutants and measuring the fraction of them killed by that test suite. This fraction is called the ***mutation adequacy score*** (or just ***mutation score***).
    - A test suite with a **higher score is better**.

Mutation testing: interpreting the score

- Is it a good idea to ask your LLM to try to write a test suite that achieves 100% mutation score? Why or why not?
  - ***Equivalent Mutant Problem***: semantically-equivalent mutants cannot be detected by any test suite and dilute the score
    - Checking if two programs are semantically-equivalent is **undecidable**, so this problem is not solvable in the limit
- As a result:
  - Mutation scores are not comparable across programs
  - There is no perfect mutation score target

Mutation testing: interpreting the score

- Is it a good idea to ask your LLM to try to write a test suite that achieves 100% mutation score? Why or why not?

Mutation testing: interpreting the score

- Is it a good idea to ask your LLM to try to write a test suite that achieves 100% mutation score? Why or why not?
  - ***Equivalent Mutant Problem***: semantically-equivalent mutants cannot be detected by any test suite and dilute the score

Mutation testing: interpreting the score

- Is it a good idea to ask your LLM to try to write a test suite that achieves 100% mutation score? Why or why not?
  - ***Equivalent Mutant Problem***: semantically-equivalent mutants cannot be detected by any test suite and dilute the score
    - Checking if two programs are semantically-equivalent is **undecidable**, so this problem is not solvable in the limit

Mutation testing: interpreting the score

- Is it a good idea to ask your LLM to try to write a test suite that achieves 100% mutation score? Why or why not?
  - ***Equivalent Mutant Problem***: semantically-equivalent mutants cannot be detected by any test suite and dilute the score
    - Checking if two programs are semantically-equivalent is **undecidable**, so this problem is not solvable in the limit
- As a result:
  - Mutation scores are not comparable across programs
  - There is no perfect mutation score target

In-class Activity: Red-Team vs Blue-Team

- Earlier, you were assigned to an **offensive** or **defensive** team
  - Now, you should find your counterpart team (w/ same letter)

In-class Activity: Red-Team vs Blue-Team

- Earlier, you were assigned to an **offensive** or **defensive** team
  - Now, you should find your counterpart team (w/ same letter)
- **Defensive** teams will try to improve their course project's test suite using the techniques we discussed in class + LLMs
  - "Stryker" is the usual mutation testing tool for JavaScript

In-class Activity: Red-Team vs Blue-Team

- Earlier, you were assigned to an **offensive** or **defensive** team
  - Now, you should find your counterpart team (w/ same letter)
- **Defensive** teams will try to improve their course project's test suite using the techniques we discussed in class + LLMs
  - "Stryker" is the usual mutation testing tool for JavaScript
- **Offensive** teams will try to find bugs in their partner defensive team's project by generating new test cases (with any method)
  - Defensive team is responsible for helping you set up their project

In-class Activity: Red-Team vs Blue-Team

- Earlier, you were assigned to an **offensive** or **defensive** team
  - Now, you should find your counterpart team (w/ same letter)
- **Defensive** teams will try to improve their course project's test suite using the techniques we discussed in class + LLMs
  - "Stryker" is the usual mutation testing tool for JavaScript
- **Offensive** teams will try to find bugs in their partner defensive team's project by generating new test cases (with any method)
  - Defensive team is responsible for helping you set up their project
- **Mild contest**: who can find more bugs by the end of class?

Wrapup and Reminders

- P5 due Sunday