VANDERBILT
School of Engineering

Final Exam Review

CS 4278/5278: Principles of Software Engineering

Zihan Fang
Graduate Teaching Assistant
Zihan Fang@vanderbilt.edu

HW6 b

- Sunday April 21
- The grace period and late policy do not apply

T 04/16/24	**Final Exam Review.**
TR 04/18/24	**Final Exam (Exam2).**
04/21/24	*(None; this is a sunday)*

Exam 2

- **Thursday April 18**
- Class Time (75 min)
    - 1:15 PM - 2:30 PM
    - FGH 134
- TA-Proctored
- Paper-based, written exam

T 04/16/24	**Final Exam Review.**
TR 04/18/24	**Final Exam (Exam2).**
04/21/24	*(None; this is a sunday)*

VANDERBILT
School *of* Engineering

Exam 2

- **Thursday April 18**
- Class Time (75 min)
  - 1:15 PM - 2:30 PM
  - FGH 134
- TA-Proctored
- Paper-based, written exam

T 04/16/24	**Final Exam Review.**
TR 04/18/24	**Final Exam (Exam2).**
04/21/24	*(None; this is a sunday)*

NO ChatGPT

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

Exam Structure

- 100 points in total + 5 extra credits
- 5 multipart questions (10-20% are covered in Exam 1)
- 1 multipart bonus
- Short answer, answer bank, fill in the blank
- Open-book, open-notes, open-internet

Exam Topics

- Delta Debugging
- Requirements and Specifications
- Maintainability and Productivity
- Fault Localization
- Automated Programming Repair
- Profiling

Delta Debugging

- Delta debugging is an **automated debugging approach** that finds a one-**minimal interesting subset** of a given set.
- Delta debugging is based on **divide and conquer** and relies on critical **assumptions** (monotonicity, unambiguity, and consistency).
- It can be used to find which code changes cause a bug, to minimize failure inducing inputs, and even to find harmful thread schedules.

Delta Debugging

Remember the three main assumptions around Delta Debugging…

- Monotonicity - if X is interesting, set of X & anything is interesting
- Unambiguity - if X & Y are interesting, intersection of X & Y is interesting
- Consistency - X is either interesting or not interesting

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, …, 8}

1	2	3	4	5	6	7	8	Interesting?

Example: Use DD to find the smallest interesting subset of {1, …, 8}

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, …, 8}

1	2	3	4	5	6	7	8	Interesting?
1	2	3	4					
				5	6	7	8	

First Step:
Partition C = {1, …, 8} into
P1 = {1, …, 4} and P2 = {5, …, 8}

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, ..., 8}

1	2	3	4	5	6	7	8	Interesting?
1	2	3	4					???
				5	6	7	8	???

Second Step:
Test P1 and P2

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, …, 8}

1	2	3	4	5	6	7	8	Interesting?
1	2	3	4					No
				5	6	7	8	No

Interference! Sub-Step:
Find minimal subset D1 of P1
such that Interesting(D1 + P2)

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, …, 8}

1	2	3	4	5	6	7	8	Interesting?
1	2	3	4					No
				5	6	7	8	No
1	2			5	6	7	8	???

Interference! Sub-Step:
Find minimal subset D1 of P1
such that Interesting(D1 + P2)

VANDERBILT School of Engineering

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, …, 8}

1	2	3	4	5	6	7	8	Interesting?
1	2	3	4					No
				5	6	7	8	No
1	2			5	6	7	8	No
		3	4	5	6	7	8	Yes

Interference! Sub-Step:
Find minimal subset D1 of P1
such that Interesting(D1 + P2)

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, …, 8}

1	2	3	4	5	6	7	8	Interesting?
1	2	3	4					No
				5	6	7	8	No
1	2			5	6	7	8	No
		3	4	5	6	7	8	Yes
		3		5	6	7	8	Yes

D1 = {3}

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, …, 8}

1	2	3	4	5	6	7	8	Interesting?
1	2	3	4					No
				5	6	7	8	No
1	2			5	6	7	8	No
		3	4	5	6	7	8	Yes
		3		5	6	7	8	Yes
1	2	3	4	5	6			Yes

D1 = {3}

Now find D2!

Delta Debugging

Example: {3,6} Is Smallest Interesting Subset of {1, …, 8}

1	2	3	4	5	6	7	8	Interesting?
1	2	3	4					No
				5	6	7	8	No
1	2			5	6	7	8	No
		3	4	5	6	7	8	Yes
		3		5	6	7	8	Yes
1	2	3	4	5	6			Yes
1	2	3	4	5				No
1	2	3	4		6			Yes

D1 = {3}
D2 = {6}

Requirements

- Requirements say what the system will do, not how it will do it
- System requirements: relationships between monitored and controlled variables
- Software requirements: relationship between inputs and outputs
- Produce formal software requirement models:
  - Functional requirements
  - Non-functional requirements (quality requirements)

Readability

Readability is a human judgment of how easy a text is to understand

- Avoid long lines
- Avoid having many different identifiers in the same region of code
- Do include comments
- Fully blank lines may matter more than indention

Code Inspection and the Brain

- Comprehending code is where developers spend most time
- What makes code easy to read? Should we ask programmers?
- Self-reporting is unreliable
  - High variability and low mean validity

Code Inspection and the Brain

Summary of Techniques:

- fMRI
- fNIRS
- Eye tracking
- Smartwatch data
- Surveys
- Interviews

Productivity

- Experiment with system response time
  - Short term mental memory buffer can be disrupted by increased system response time
  - Faster response time enabled significant performance enhancement
  - Cost of upgrading a processor can be more than justified by savings in user time
- "Programming speed" - higher-order language, less CPU time, faster coding
- "Program economy" - faster running programs, experience, lower-level language

Productivity

- Main idea: programming speed (associated with a higher-order language, faster coding, less CPU time) is a commonly mistaken belief
- Using abstraction is the real path to success
- Can get abstraction through language, or other avenues - the ideal of abstraction is the insight
- Abstraction can take years, but that is the true limitation to productivity

Patterns & Anti-Patterns

- Patterns: reusable solutions to common software problems
- Structural
  - Adapter
- Creational
  - Named constructor, factory, abstract factory, singleton
- Behavioral
  - Iterator, observer, template

Patterns & Anti-Patterns

- Anti-pattern: an ineffective solution to a problem
- Psychology: Hick's Law - increasing # of choices increases decision time logarithmically
    - Application to menu and UI design

Fault Localization

- Fault Localization: identifying lines implicated in a bug. Humans are better at localizing some types of bugs than others.
- Debugger: **single-stepping** through the program and inspecting variable values.
- Automatic tools can help with the dynamic analyses of fault localization and profiling

Debugger

- What is a debugger?
  - Can operate on source code or assembly code
  - Inspect the values of registers, memory
  - Key Features
    - Attach to process
    - Single-stepping
    - Breakpoints
    - Conditional Breakpoints
    - Watchpoints

Fault Localization Tools

- Spectrum-Based Fault Localization
  - Dynamic Analysis
  - Comparing statements covered on failing test cases to statements covered on passing test cases
- Coverage-Based Fault Localization

Statement	3,3,5	1,2,3	3,2,1	3,2,1	5,5,5	2,1,3
int m;						
m = z;						
if (y < z)						
if (x < y)						
m = y;						
else if (x<z)						
m = y; // bug						
else						
if (x > y)						
m = y;						
else if (x>z)						
m = x;						
return m;						
	Pass	Pass	Pass	Pass	Pass	Fail 39

Automatic Program Repair

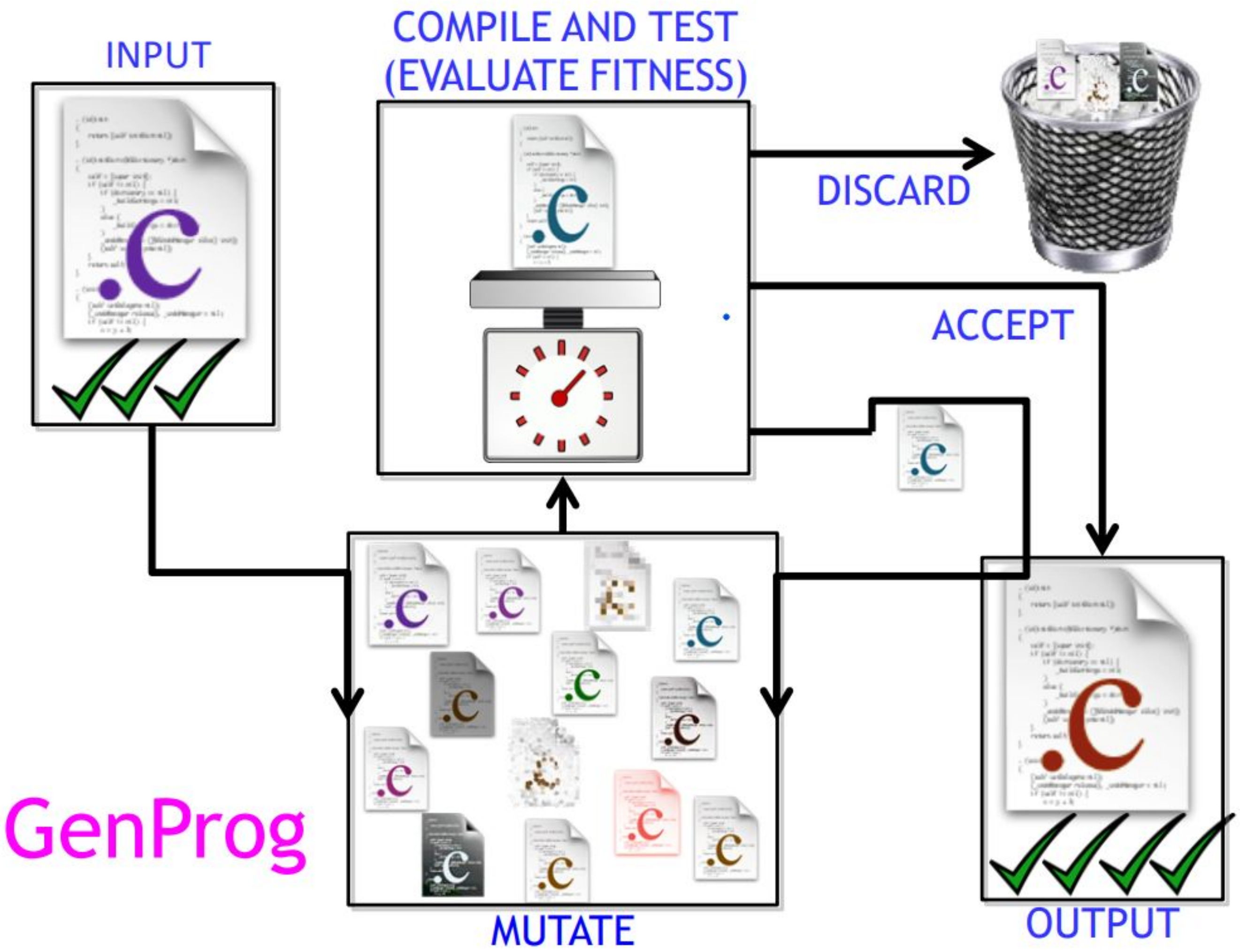
- Anyone can submit a bug report in “bug bounty” programs at major software companies
- More economical to pay strangers to submit defect reports
- Only 38% are true positives, but that’s still a lot of bugs
- We have more bugs than time to repair them

Automatic Program Repair

- Can use strategies and techniques learned in this class to find evidence of and fix existing bugs
- Fault localization, mutation, testing to find/fix bugs
- A patch might contain extraneous edits (use delta debugging to minimize)
- Each repair has to pass the whole test suite
- Can use static analysis to prevent testing “duplicates” aka equivalent patches

Automatic Program Repair

- Ideally…
  - Mutation testing takes a program that passes all tests, and human mistake-based mutants (that aren't equivalent) must fail at least one test
  - Program repair takes a program that fails test suite, requires that one mutant (based on human repairs from fault localization) only passes all tests


INPUT
COMPILE AND TEST
(EVALUATE FITNESS)
DISCARD
ACCEPT
GenProg
MUTATE
OUTPUT

Automatic Program Repair

- APR is good at fixing lots of bugs
  - Typically require small changes
  - Changes typically have to be AST modifications
- APR isn't so good at other types of bugs (yet)
  - Particular values being off
  - Bugs that require human expertise

Profiling

- A profiler is a performance analysis tool that measures the frequency and duration of function calls as a program runs.
- A flat profile computes the average call times for functions but does not break times down based on context.
- A call-graph profile computes call times for functions and also the call-chains involved
- E.x., event-based profiling, statistical profiling

Profiling

- Event-Based Profiling
  - Interpreted languages provide special hooks for profiling
    - Java: JVM-Profile Interface, JVM API
    - Python: sys.set_profile() module
    - Ruby: profile.rb, etc.
- Statistical Profiling
  - You can arrange for the operating system to send you a signal every X seconds
  - In the signal handler you determine the value of the target program counter
  - And append it to a growing list file, this is sampling
  - Later, you use debug information from the compiler to map the PC values to procedure names
  - Sum up to get amount of time in each procedure

Please complete the course evaluation!