Final Exam Review

CS 4278/5278: Principles of Software Engineering

Skyler Grandel
Graduate Teaching Assistant
skyler.h.grandel@vanderbilt.edu

Department of Computer Science

VANDERBILT
SCHOOL OF ENGINEERING

Come Participate in Our Study!

Software Comment Analysis

```
/**
 * What does this comment say about its code?
 *
 * @Task - Identify the purpose of a C/C++ comment
 *         and indicate if it is relevant
 *
 * @Duration - 45 to 60 minutes
 *
 * @Compensation - $20
 *
 * @Location - Online
 *
 * @Contact - Email skyler.h.grandel@vanderbilt.edu
 */
```

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

And the problems that delta debugging seeks to solve are simplifying, isolating, and identifying failure-inducing components

Fault Localization Overview

- Debugger: **single-stepping** through the program and inspecting variable values.
- Fault Localization: identifying lines implicated in a bug. Humans are better at localizing some types of bugs than others.
- 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

Signals

- Asynchronous notification sent to a process about an event
- Signal handler: a procedure that will be executed when the signal occurs.
  - vulnerable to race conditions

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

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

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.
- You register a function that will get called whenever the target program calls a method, loads a class, allocates an object, etc.
  - cf. “signal handler”

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

Sampling Analysis

- Advantages
  - Simple and cheap – the instrumentation is unlikely to disturb the program
  - No big slowdown
- Disadvantages
  - Can completely miss periodic behavior (e.g., you sample every k seconds but do a network send at times 0.5 + nk seconds)
  - High error rate

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

- Structural patterns
  - Simplify relationships between entities by creating interfaces
  - Hides implementation details
  - Example: Adapter pattern
    - Common adapters: Stack, fstream in C++, autograder

Patterns & Anti-Patterns

- Creational patterns
  - Control object creation
  - Factory pattern
    - Purpose is to create objects without having return type specify exact subclass
  - Abstract Factory pattern
    - Encapsulates group of factories, without specifying concrete classes
  - Singleton pattern
    - Only allows one instance of a class, provides global access to it

Patterns & Anti-Patterns

- Behavioral patterns
  - Support communication patterns between objects
  - Iterator
    - Traversal of containers irrespective of implementation
  - Observer
    - Dependent objects are notified upon a state change
    - One-to-many: one object updates, many are notified

Patterns & Anti-Patterns

- Behavioral patterns
  - Support communication patterns between objects
  - Template method
    - Deferring some portion of work to subclass by allowing it to override the “default” virtual methods
  - Publish-Subscribe
    - Message senders publish on topics and receivers subscribe to topics
    - Neither pubs nor subs need to know about each other

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

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 (3 of top 4 self-reported features are irrelevant)
  - High variability and low mean validity

Code Inspection and the Brain

- Brain uses energy, energy transported into the brain with oxygen in the blood
- Can use differing electromagnetic properties of oxygen-rich or -poor blood to tell where brain function is occurring with magnetic resonance (MR) scanner
- Functional magnetic resonance imaging (fMRI) is non-invasive way to probe for cognitive function in this way

Code Inspection and the Brain

- So what works?
- Top-down comprehension - experience, expectation, and semantic cues (known as beacons) guide understanding
  - Plans
- Bottom-up comprehension - meaning is obtained from every individual statement, then put all together into a larger, holistic understanding
  - Semantic clunking

Code Inspection and the Brain

- Comprehension with semantic cues requires less cognitive effort than bottom-up comprehension
- When writing comments or naming identifiers, do so with the “why” in mind
- Code writing requires more activity in parts of the brain associated with top-down control, prose writing, planning, etc.

Code Inspection and the Brain

- Using fMRI analysis, hard to process data
- Can classify which task a participant is undertaking only based on brain activity
- Means that Code Review, Code Comprehension, and Prose Review all have distinct neural representations
- As proficiency in coding increases, neural representations of code and prose are less differentiable (still distinct, though)

Code Inspection and the Brain

Summary of Techniques:

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

Automatic Program Repair

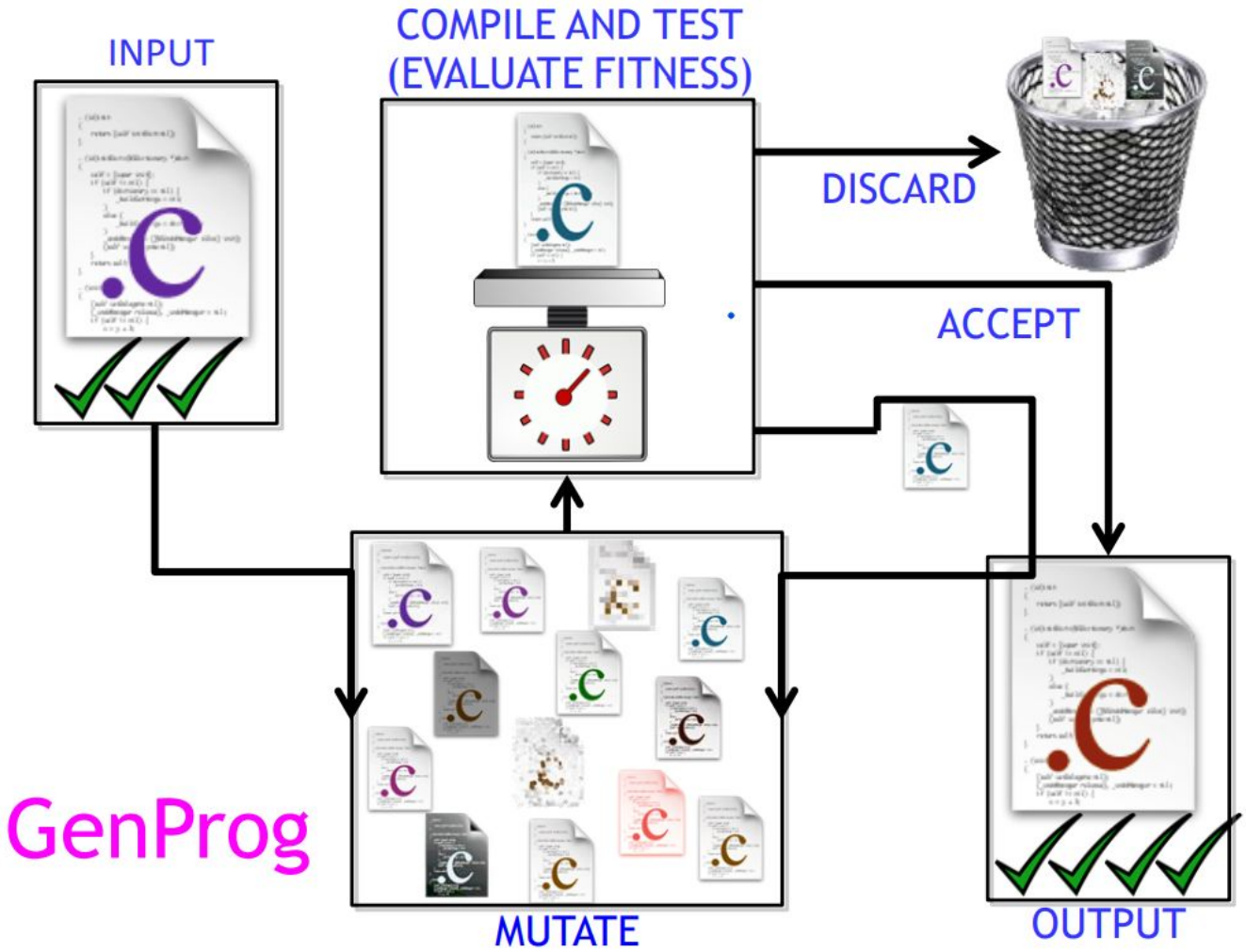
- Anyone can submit a bug report in “bug bounty” programs at major software companies
- More economical to pay strangers to submit defect reports
- Only 38% are true positives, but that’s still a lot of bugs
- We have more bugs than time to repair them

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

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

- Experts just solve problems in one step - quicker
- Novices focus more on surface features
- Experts focus on underlying principles
- With learning, shift in how knowledge is organized (from surface to principles)
- Improving how one learns would be done by identifying available knowledge and manipulating or working off of that

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