CS 520
Theory and Practice of Software Engineering
Fall 2022

In-class exercise 3: Debugging

November 1, 2022
Ways to get your code right

• Validation (e.g., code reviews, testing, model checking)
  – Purpose is to uncover problems and increase confidence

• Debugging
  – Finding out why a program is not functioning as intended

• Defensive programming
  – Programming with validation and debugging in mind

• Validation ≠ debugging
  – Validation: Reveals existence of problem
  – Debugging: Pinpoints location + cause of problem
Defenses in depth

1. Make errors impossible
   – e.g., Java makes memory overwrite bugs impossible

2. Don’t introduce defects
   – Correctness: get things right the first time

3. Make errors immediately visible: Local visibility of errors: best to fail immediately
   – e.g., assertions to check rep(representation) invariants
Some Fault Localization Techniques

• Manual inspection
• Assertions
• IDE debugger
• Delta debugging
• ...

Delta Debugging is a methodology to automate the debugging of programs using a scientific approach of hypothesis-trial-result loop. This methodology was first developed by Andreas Zeller of the Saarland University in 1999.\[1\]

In practice, the Delta Debugging algorithm builds on unit testing to isolate failure causes automatically - by systematically narrowing down failure-inducing circumstances until a minimal set remains. For example, if you can supply a test case that will produce the bug you are looking for, then you can feed that to the Delta Debugging algorithm, which will then simply try to trim useless functions and lines of code that are not needed to reproduce the bug, until a 1-minimal program is found.

Delta Debugging has been applied to isolate failure-inducing program input (e.g. an HTML page that makes a Web browser fail), failure-inducing user interaction (e.g. the keystrokes that make a program crash), or failure-inducing changes to the program code (e.g. after a failing regression test).

Later, some software development tools have been inspired by Delta Debugging, such as the bisect commands of revision control systems (e.g., git-bisect, svn-bisect, hg-bisect, etc.), which, instead of working on the program’s code, apply the delta debugging methodology on the code history by comparing various versions until the faulty change is found.

Delta debugging

Delta Debugging is a methodology to automate the debugging of programs using a scientific approach of hypothesis-trial-result loop. This methodology was first developed by Andreas Zeller of the Saarland University in 1999.[1]

In practice, the Delta Debugging algorithm builds on unit testing to isolate failure causes automatically - by systematically narrowing down failure-inducing circumstances until a minimal set remains. For example, if you can supply a test case that will produce the bug you are looking for, then you can feed that to the Delta Debugging algorithm, which will then simply try to trim useless functions and lines of code that are not needed to reproduce the bug, until a 1-minimal program is found.

Delta Debugging has been applied to isolate failure-inducing program input (e.g. an HTML page that makes a Web browser fail), failure-inducing user interaction (e.g. the keystrokes that make a program crash), or failure-inducing changes to the program code (e.g. after a failing regression test).

Later, some software development tools have been inspired by Delta Debugging, such as the bisect commands of revision control systems (e.g., git-bisect, svn-bisect, hg-bisect, etc.), which, instead of working on the program's code, apply the delta debugging methodology on the code history by comparing various versions until the faulty change is found.

Group selection

• Form 2-, 3-, or 4-person teams
  – Use Moodle to self-select a team; open until today at noon

• Due: Tuesday November 8, 11:59 PM
Set up

1. Check if you have git (version >= 2.7.4), ant, and java (version >= 8)

2. Clone the basic-stats git repository:

   git clone https://github.com/LASER-UMASS/basic-stats basic-stats

3. Will use the git bisect command:

   https://git-scm.com/docs/git-bisect
Basic stats app (HEAD or main)

• src/
  ○ BasicStats.java
  ○ Controllers
  ○ Models
  ○ Views
• Test/
  ○ BasicStatsTest.java // ONE test case fails
Basic stats app (v1.0.0)

• src
  ○ BasicStats.java
  ○ NotSoGood.java

• test
  ○ BasicStatsTest.java // ALL test cases pass
NAME

git-bisect - Use binary search to find the commit that introduced a bug

SYNOPSIS

```bash
git bisect <subcommand> <options>
```

DESCRIPTION

The command takes various subcommands, and different options depending on the subcommand:

```bash
git bisect start [--term={new,bad}=<term> --term={old,good}=<term>]
                   [--no-checkout] [--first-parent] [bad | good]... [--] [...]
       <paths>
git bisect (bad|new|<term-new>) [...]
git bisect (good|old|<term-old>) [...]
git bisect terms [--term-good | --term-bad]
git bisect skip [(<rev>|<range>)...]
git bisect reset [<commit>]
git bisect (visualize|view)
git bisect replay <logfile>
git bisect log
git bisect run <cmd>...
git bisect help
```
Git bisect command (cont.)

- **GOOD**: All test cases pass

- **BAD**: One or more test cases fail