How do we debug?

Regression testing

- Whenever you find and fix a bug
  - Add a test for it
  - Re-run all your tests
- Why this is a good idea
  - Often reintroduce old bugs while fixing new ones
  - Helps to populate test suite with good tests
  - If a bug happened once, it could well happen again
- Run regression tests as frequently as you can afford to
  - Automate process
  - Make concise test sets, with few superfluous tests

Logging events

- Often you would like to have some indication of past when a check fails
- Design a logging infrastructure
  - Dump events to a file (strings)
  - Events have consistent format to enable efficient searches
  - Sometimes (usually for timing reasons) must keep log in memory, not on disk
  - Circular logs to avoid resource exhaustion
- Important in debugging in customer environments
  - May not have access to the customer use
  - Only the log is available
  - Information on the log to help reproduce the bug

Last resort: debugging

- Bugs happen
  - Industry average: 10 bugs per 1000 lines of code (“kloc”)
- Bugs that are not immediately localizable happen
  - Found during integration testing
  - Or reported by user

- step 1 – Clarify symptom
- step 2 – Find and understand cause, create test
- step 3 – Fix
- step 4 – Rerun all tests

Kinds of bugs

- Quick, easy bugs (few minutes)
- Medium bugs (hours)
- Hard bugs (small number of days)
- Really bad bugs (many days to never)

- Look for bugs in this order!
- Different debugging strategies for each

Finding Easy Bugs

- Hope for a quick bug, take a first quick shot
  - Look at backtrace in the debugger
  - Look at code where you think there might be a problem, maybe use a debugger or a few print statements in
  - Try to get lucky
- Make the first shot quick! Don’t get sucked in!

What's next?

- Look for medium bug with next shot
  - Use print statements
  - Design an organized print strategy
  - Legible, easy to read error messages
- Make the medium shot medium! Don't get sucked in!

Tricks for hard bugs

- Rebuild system from scratch and reboot
- Explain bug to a friend
- Make sure it is a bug – program may be working correctly and you don't realize it!
- Minimize input required to exercise bug
- Add checks to program
  - Minimize distance between error and detection
  - Use binary search to narrow down possible locations
- Use logs to record events in history

Delta debugging

- Find the smallest input that causes the bug.

- Imagine a long xml file that crashes the browser. How would you debug it?

Reducing input size example

boolean substr(String s, String b)

returns false for

s = “The wworld is ggreat! Liffe is wwonderful! I am so vvery happy all of the ttime!”

b = “very happy”

even though “very happy” is a substring of s

Wrong approach: try to trace the execution of substr for this case

Right approach: try to reduce the size of the test case

Reducing input size example

substr("I am so vvery happy all of the ttime!", "very happy") == false
substr("very happy all of the ttime!", "very happy") == true
substr("I am so vvery happy", "very happy") == false
substr("I am so vvery happy", "happy") == true
substr("I am so vvery happy", "very") == false
substr("I am so vvery happy", "ve") == false
substr("vvery happy", "ve") == false
substr("vvery happy", "v") == true
substr("vvery", "ve") == false
substr("vve", "ve") == false
substr("ve", "ve") == true

General strategy: simplify

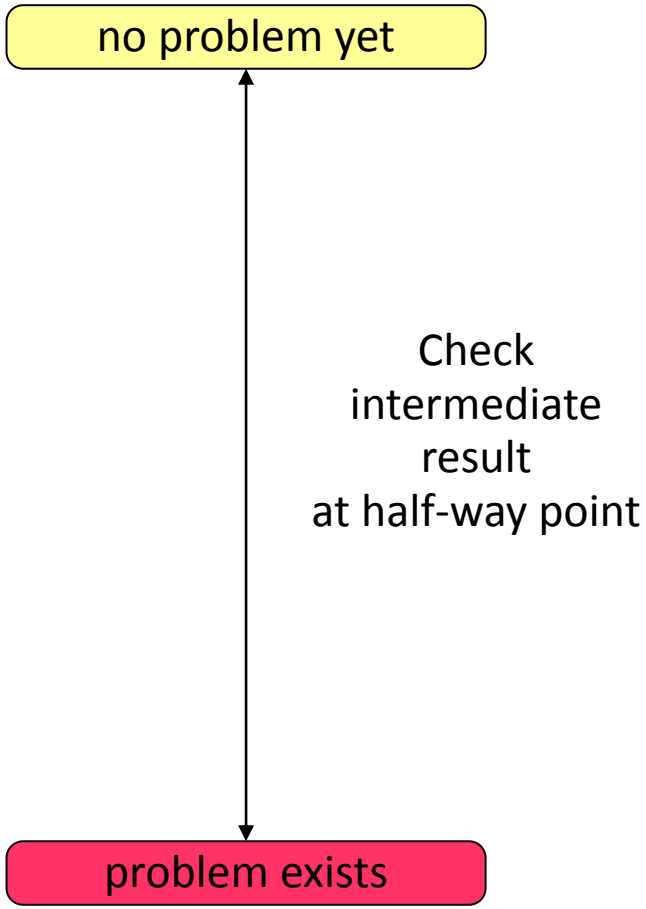
- In general: find simplest input that will provoke bug
  - Usually not the input that revealed existence of the bug
- Start with data that revealed bug
  - Keep paring it down (binary search can help)
  - Often leads directly to an understanding of the cause
- When not dealing with simple method calls
  - Think of "test input" as the set of steps needed to reliably trigger the bug
  - Same basic idea

Localizing a bug

- Take advantage of modularity
  - Start with everything, take away pieces until bug goes
  - Start with nothing, add pieces back in until bug appears
- Take advantage of modular reasoning
  - Trace through program, viewing intermediate results
- Can use binary search to speed things up
  - Bug happens somewhere between first and last statement
  - So can do binary search on that ordered set of statements

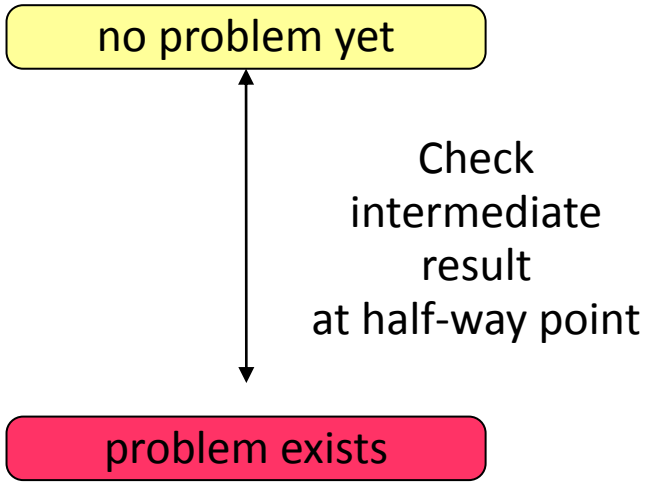
binary search on buggy code

```
public class MotionDetector {
   private boolean first = true;
   private Matrix prev = new Matrix();

   public Point apply(Matrix current) {
      if (first) {
         prev = current;
      }
      Matrix motion = new Matrix();
      getDifference(prev,current,motion);
      applyThreshold(motion,motion,10);
      labelImage(motion,motion);
      Hist hist = getHistogram(motion);
      int top = hist.getMostFrequent();
      applyThreshold(motion,motion,top,top);
      Point result = getCentroid(motion);
      prev.copy(current);
      return result;
   }
}
```



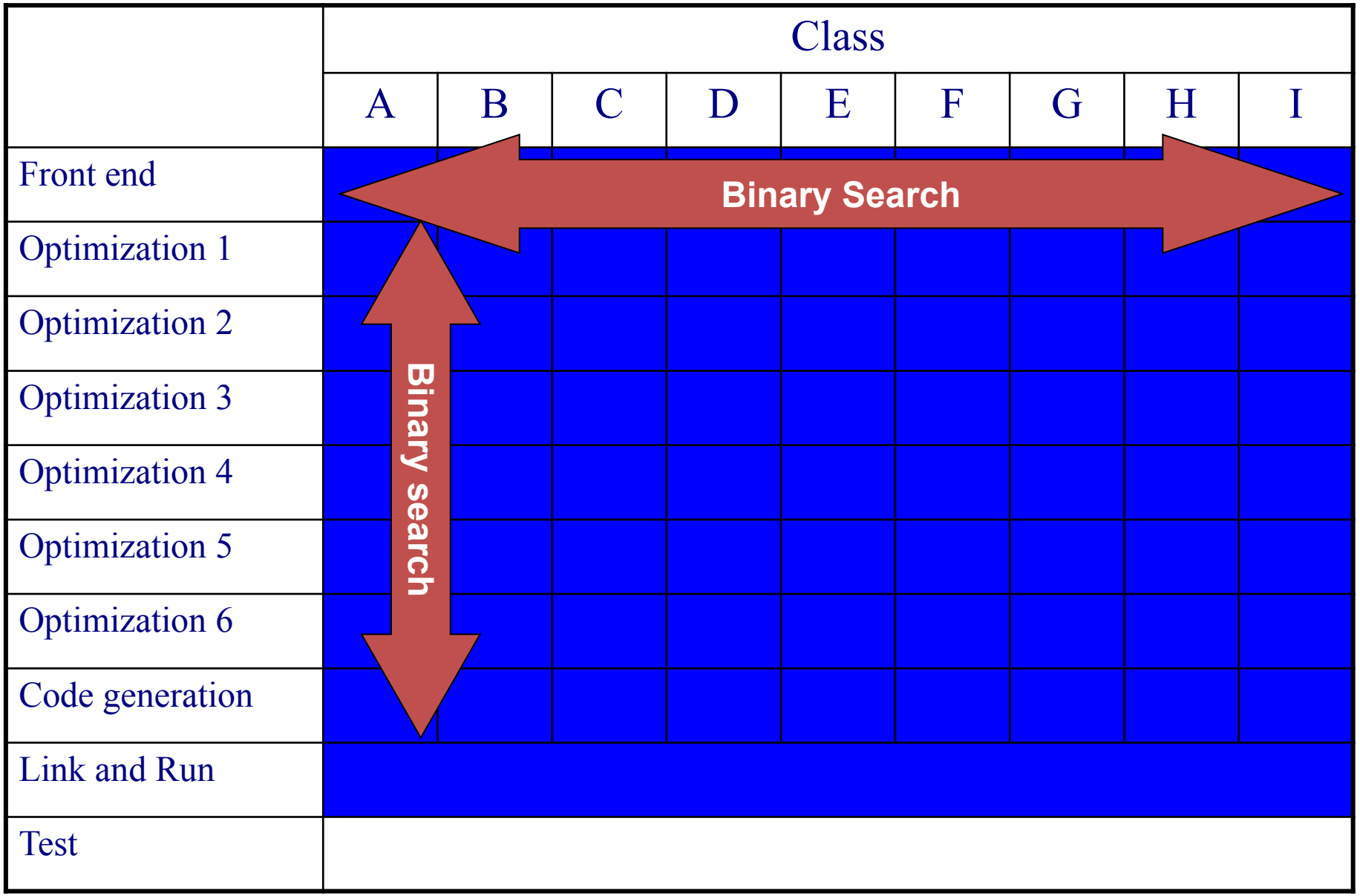
binary search on buggy code

```
public class MotionDetector {
   private boolean first = true;
   private Matrix prev = new Matrix();

   public Point apply(Matrix current) {
      if (first) {
         prev = current;
      }
      Matrix motion = new Matrix();
      getDifference(prev,current,motion);
      applyThreshold(motion,motion,10);
      labelImage(motion,motion);
      Hist hist = getHistogram(motion);
      int top = hist.getMostFrequent();
      applyThreshold(motion,motion,top,top);
      Point result = getCentroid(motion);
      prev.copy(current);
      return result;
   }
}
```



Quickly home in
on bug in O(log n) time
by repeated subdivision

Binary Search in a Compiler



Binary Search in a Compiler

	Class								
	A	B	C	D	E	F	G	H	I
Front end	■	■	■	■	■	■	■	■	■
Optimization 1									
Optimization 2	■	■				■			
Optimization 3									
Optimization 4									
Optimization 5									
Optimization 6	■	■				■			
Code generation	■	■	■	■	■	■	■	■	■
Link and Run	■	■	■	■	■	■	■	■	■
Test									

Heisenbugs

- Sequential, deterministic program – bug is repeatable
- But the real world is not that nice…
  - Continuous input/environment changes
  - Timing dependencies
  - Concurrency and parallelism

- Bug occurs randomly

- Hard to reproduce
  - Use of debugger or assertions → bug goes away
  - Only happens when under heavy load
  - Only happens once in a while

Debugging in harsh environments

- Harsh environments
  - Bug is nondeterministic, difficult to reproduce
  - Can't print or use debugger
  - Can't change timing of program (or bug has to do with timing)
- Build an event log (circular buffer)
- Log events during execution of program as it runs at speed
- When detect error, stop program and examine logs

Where is the bug?

- The bug is not where you think it is
  - Ask yourself where it cannot be; explain why
- Look for stupid mistakes first, e.g.,
  - Reversed order of arguments: Collections.copy(src,dest)
  - Spelling of identifiers: int hashcode()

    `@override` can help catch method name typos
  - Same object vs. equal: a == b versus a.equals(b)
  - Failure to reinitialize a variable
  - Deep vs. shallow copy
- Make sure that you have correct source code
  - Recompile everything

When the going gets tough

- Reconsider assumptions
  - Has the OS changed? Is there room on the hard drive?
  - Debug the code, not the comments
- Start documenting your system
  - Gives a fresh angle, and highlights area of confusion
- Get help
  - We all develop blind spots
  - Explaining the problem often helps
- Walk away
  - Trade latency for efficiency – **sleep**!
  - One good reason to start early

Detecting bugs in the real world

- Real systems are…
  - Large and complex
  - Collection of modules, written by multiple people
  - Complex input
  - Many external interactions
  - Non-deterministic
- Replication can be an issue
  - Infrequent bug
  - Instrumentation eliminates the bug
- Bugs cross abstraction barriers
- Large time lag from corruption to detection

Key Concepts in Review

- Testing and debugging are different
  - Testing reveals existence of bugs
  - Debugging pinpoints location of bugs
- Goal is to get program to work
  - Not to find bugs
- Debugging should be a systematic process
  - Use the scientific method
- It is important to understand source of bugs
  (to decide on appropriate repair)

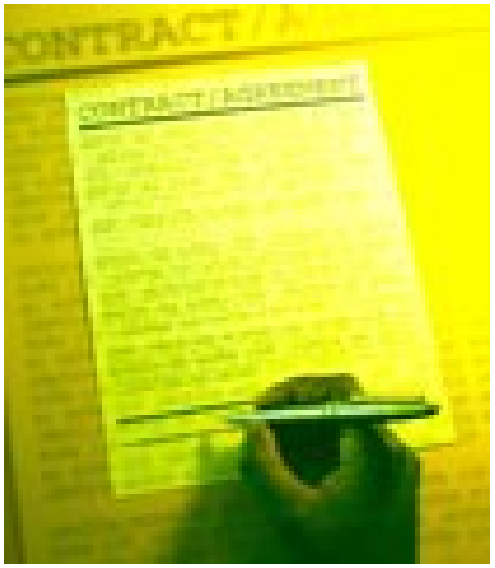
Intellectual Property

Common forms of IP

- Patents
- Copyrights
- Trade secrets
- Trademarks
- Contracts/Licenses

Contracts/Licenses

- You can make anything work if you have agreement by all parties involved
- Protections, exclusions, requirements, terms, and costs must all be *explicitly* defined as part of the contract
- Examples:
  - License agreements
  - Vendor agreements
  - Non-disclosure agreements
  - Employee contracts



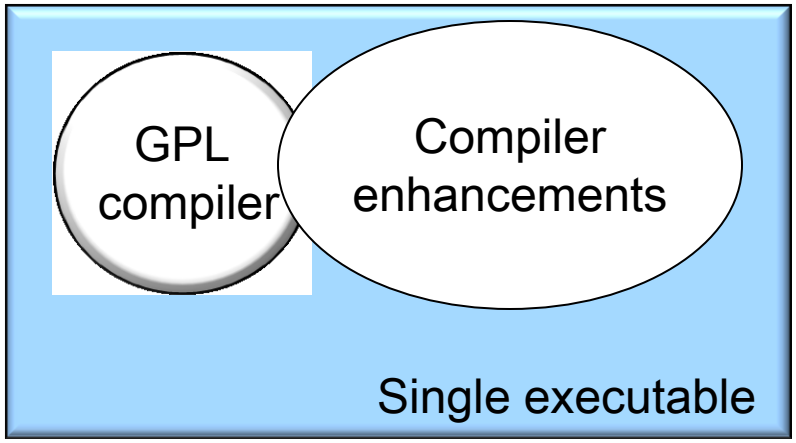
Free Software Foundation

The Free Software Foundation (FSF), 1985–
dedicated to promoting computer users' rights to
use, study, copy, modify, and redistribute computer programs

FSF promotes the freedom to

1. run the program, for any purpose
2. study how the program works, and adapt it to your needs
3. redistribute copies so you can help your neighbor
4. improve the program, and release your improvements to the public, so that the whole community benefits

GPL – GNU General Public License

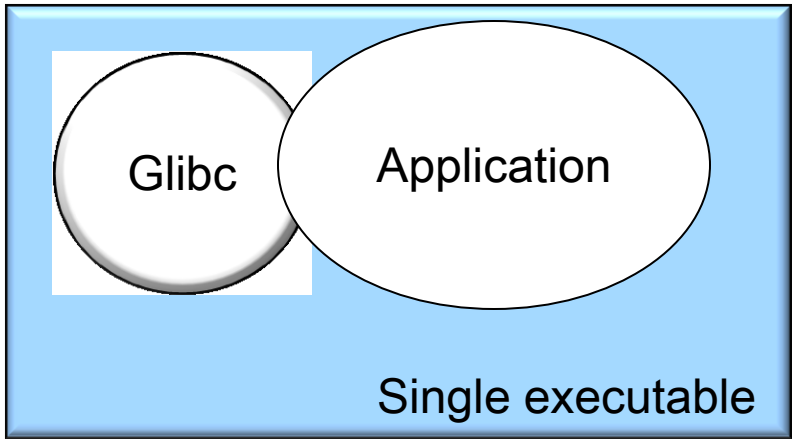


- Software licensed under the GPL is open source that must be made available with the product release

- GPL requires that all code that is “affected” by GPL code must also be distributed under the GPL. “affected” is loosely equated to, part of the same functional unit

- poisonous license, copyleft license

LGPL



- LGPL is referred to as the “Lesser” GPL or “Library” GPL
- Applications may link to LGPL code without having to take on the LGPL license

There are lots of other licenses

- Mozilla Public License (bugzilla) – type of copyleft
- MIT License (ruby on rails) - permissive FOSS
- BSD License (Trac, PostgreSQL) - permissive FOSS
- MySQL – GPL or MySQL commercial license ($)

Microsoft takes on the free world

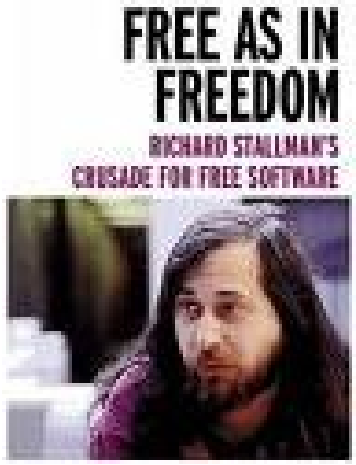
Microsoft alleges that FOSS infringes on 235 of its patents
It wants royalties from distributors and users

- the Linux kernel violates 42 Microsoft patents
- the Linux user interface, design elements infringe on 65 patents
- OpenOffice.org infringes on 45
- another 83 are infringed on in other FOSS programs

"What's fair is fair," Ballmer told *Fortune*.
"We live in a world where we honor, and
support the honoring of, intellectual property."

http://money.cnn.com/magazines/fortune/fortune_archive/2007/05/28/100033867/

And the FOSS response is?



- FOSS legal strategist is “uncowed”
  - The action is in tight qualitative analysis of individual situations
  - Patents can be invalidated on numerous grounds
  - Others can be invented around
  - Supreme Court stated that patents have been issued too readily for the past two decades and lots are probably invalid
  - Corporate patrons and allies

What's your opinion?

- "Patents and the open-source movement get along awkwardly at best.
  Patent law gives proprietary, exclusive rights to patent holders, but open-source programming is built on the idea of free sharing."

- Is FOSS (Free and Open Source Software) the best way to foster the advancement of science?

- Isn't it reasonable for a company to patent technology to attain / keep competitive advantage?