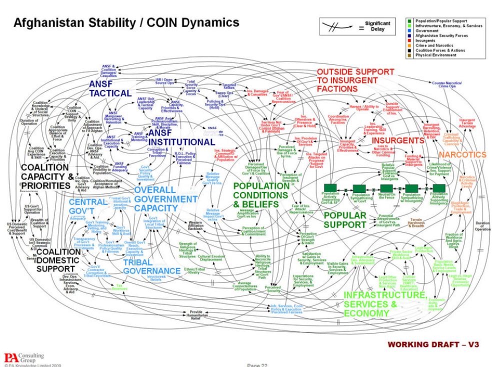
CS 521/621
Course Overview:
Static and Dynamic Analyses

Last time

What did we talk about?



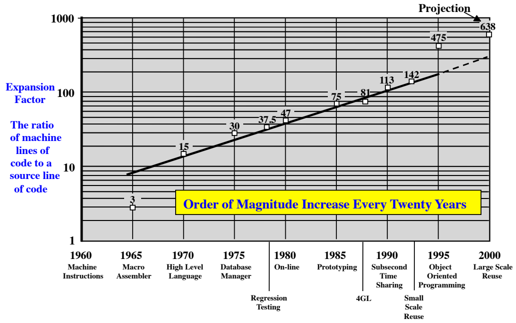
Why is it important to study software engineering?

Just like cars

- US automobile industry used to be very complacent about quality
  - lost a significant amount of market share
  - complacency about software quality could lead to the same result
- There are many recalls for automobiles
  - some fixed for free
- There are many defects in software
  - some fixed for free
  - some fixed in the the next release
    - customer paying for the upgrade

Why is analysis important?

Trends in Software Expansion (Bernstein, 1997)



Significant increase in software control

- 1960
  – 8% of F-4 Fighter capability was provided by software

- 2000
  – 85% of F-22 Fighter capability is provided by software

GAO, Report to the Committee on Armed Services, U.S. Senate, March 2004, pg. 4

Accidents

- USS Yorktown
  http://www.slothmud.org/~hayward/mic_humor/nt_navy.html
  – Suffered a systems failure when bad data was fed into its computers during maneuvers off the coast of Cape Charles,VA
  – Ship towed into the Naval base at Norfolk,VA, because a database overflow caused its propulsion system to fail
  – Took two days of pier-side maintenance to fix the problem
- Ariane Five
  http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html
  – Reused a module developed for Ariane 4, which assumed that the horizontal velocity component would not overflow a 16-bit variable
  – Not true for Ariane 5, leading to self-destruction roughly 40 seconds after launch

Any questions?

Some logistics

- 521 vs. 621
  – 621 is graduate students only
  – 521 is undergraduate or graduate

  – the material is the same, the midterm is the same, the assignments are the same
  – only three differences:
    - 621 students must do a project + 1 paper presentation
    - 521 students must do 2 paper presentations
    - Grading (scaling) is separate

TA

Ruisi Zhang

- Office hours: Tuesday 4–5PM in CS207
- ruisizhang@cs.umass.edu

Any questions?

Today's (and not only today's) plan

- Static analysis
- Dynamic analysis
- Model checking
- Mutation testing
- Bug localization
- Symbolic execution

Areas we will cover in this course

- Static analysis
- Dynamic analysis
- Model checking
- Mutation testing
- Bug localization
- Symbolic execution

areas for your projects

As we go over each topic…

- Think whether this sounds interesting
- Think about what kind of a tool you could make that uses this
- You are all programmers:
  think about things you've done while programming that were hard, and how these kinds of analysis might make it easier

Static Analysis

- Two kinds we'll consider:
  - Manual
  - Automatic

Manual Reviews

- Manual static analysis methods
  - Reviews, walkthroughs, inspections
- Most can be applied at any step in the lifecycle
- Have been shown to improve reliability, but
  - often the first thing dropped when time is tight
  - labor intensive
  - often done informally, no data/history, not repeatable

Reviews and walkthroughs

- Reviews
  - author or one reviewer leads a presentation of the artifact
  - review is driven by presentation, issues raised
- Walkthroughs
  - usually informal reviews of source code
  - step-by-step, line-by-line review

Inspections

- Software inspections
  - formal, multi-stage process
  - significant background & preparation
  - led by moderator
  - many variations of this approach

Experimental results

- software inspections have repeatedly been shown to be cost effective
- increases front-end costs
  - ~15% increase to pre-code cost
- decreases overall cost

IBM study

- Doubled number of lines of code produced per person
  - some of this due to inspection process
- Reduced faults by 2/3
- Found 60-90% of the faults
- Found faults close to when they were introduced

The sooner a fault is found the less costly it is to fix

Why are inspections effective?

- Knowing the product will be scrutinized causes developers to produce a better product (Hawthorne effect)
- Having others scrutinize a product increases the probability that faults will be found
- Walkthroughs and reviews are not as formal as inspections, but appear to also be effective
  - hard to get empirical results

What are the deficiencies?

- Tend to focus on error detection
  - what about other "ilities" -- maintainability, portability, etc?
- Not applied consistently/rigorously
  - inspection shows statistical improvement
- Human-intensive and often makes ineffective use of human resources
  - skilled software engineer reviewing coding standards, spelling, etc.
  - Lucent study: ½M LoCS added to 5M LoCS required ~1500 inspections, ~5 people/inspection
  - no automated support

Automatic static analysis

What can you tell me about this code:

```
public int square(int x) {
  return x * x;
}
```

Automatic static analysis

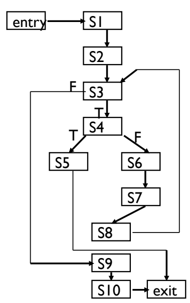
What about this code:

```
public double weird_sqrt(int x) {
  if (x > 0)
    return sqrt(x);
  else
    return 0;
}
```

Computing Control Flow Graphs (CFGs)

Procedure AVG

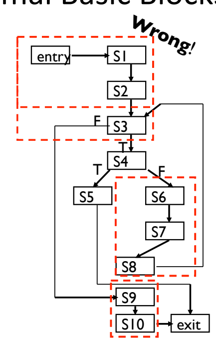
S1 count = 0
S2 fread(fptr, n)
S3 if EOF goto S11
S4 if (n >= 0) goto S7
S5 return (error)
S6 goto S9
S7 nums[count] = n
S8 count ++
S9 fread(fptr, n)
S10 goto S3
S11 avg = mean(nums,count)
S12 return(avg)



CFG with Maximal Basic Blocks

Procedure AVG

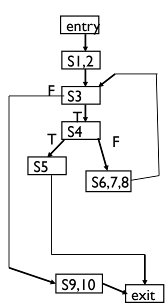
S1 count = 0
S2 fread(fptr, n)
S3 while (not EOF) do
S4 if (n < 0)
S5 return (error)
else
S6 nums[count] = n
S7 count ++
endif
S8 fread(fptr, n)
endwhile
S9 avg = mean(nums,count)
S10 return(avg)



CFG with Maximal Basic Blocks

Procedure AVG

S1 count = 0
S2 fread(fptr, n)
S3 while (not EOF) do
S4 if (n < 0)
S5 return (error)
else
S6 nums[count] = n
S7 count ++
endif
S8 fread(fptr, n)
endwhile
S9 avg = mean(nums,count)
S10 return(avg)



What about data flow?

We can do the same thing as with control flow

Uses of Data-Flow Analyses

- **Compiler Optimization**
- E.g., *Constant propagation*

a=c+10

suppose every assignment to **c** that reaches this statement assigns 5

then **a** can be replaced by 15

➡ need to know reaching definitions: which definitions of variable c reach a statement

Uses of Data-Flow Analyses

- **Software Engineering Tasks**
- E.g., Debugging

suppose that a has the incorrect value in the statement

a=c+y

➡ need data dependence information: statements that can affect the incorrect value at a given program point

Static analysis summary

- Manual or automatic
  - very different
  - manual removes bugs
- Analyze the source code to determine
  - control flow
  - data flow
- Build reachability graphs, data dependence graphs, etc.

Dynamic analysis

- Assertions
- Detecting invariants

Assertions

```
public double area(int length, int width) {
assert(length >=0);
assert(width >=0);
  return length * width;
}
```

Detecting invariants

```
public int square(int x) {
  return x * x;
}
```

Let's run the code and watch it. What can we tell about it?

Why dynamic detection?

- Is it sound?
  - If you learn a property about a program, must it be true?
- Is it complete?
  - Do you learn all properties that are true about a program?

So why dynamic detection?

- Code can be complex
  - Static analysis may not scale to large programs.
- Sometimes, logs is all you have access to
  - Not all code is open source. If you use libraries, others' code, you may only be able to observe executions.
- Fast
- Detects properties of actual usage, rather than all possible usage

What can we do with static and dynamic analyses?

- You have:
  - a program
  - some tests that pass
  - some tests that fail

What can we do with static and dynamic analyses?

- You have:
  - a program
  - some tests that pass
  - some tests that fail

What can we do statically?

Statically, we can…

- Think about the code long and hard, and fix it.
- Can we step through a failing test case? See where the code goes wrong?
  - but to automate this, we have to know where the code is "supposed" to go
- Can we reverse-engineer the conditions necessary to get to the desired result?

What can we do with static and dynamic analyses?

- You have:
  - a program
  - some tests that pass
  - some tests that fail

What can we do dynamically?

Dynamically, we can…

- Run the code and observe which lines execute when
  - lines that execute on failings tests only are more likely buggy
- We can detect code invariants and reason about the code
- We can muck with the code and see if it does any better on the tests