CSE 503: Software Engineering

Winter 2014

Lecturer: Michael Ernst

503 Software Engineering *Research*

- Not: how to write good software
  - and get a good job at Amazon/Google/Microsoft
- Research methods and ideas in SE
  - this may make you a more thoughtful developer

What does my program do?

Program analysis techniques:

- Abstract interpretation
- Type systems
- Model checking
- Analysis back-ends
- Test generation
- Dynamic analysis
- Refactoring
- Slicing
- More

Abstract interpretation (or "dataflow analysis")

- Statically (over-)estimate what the program may do at run time
- "Run" your program statically
  - Choose an abstract domain; e.g., { +, 0, - }
  - Assign semantics to operators
  - Start at beginning of program
  - Examine possible values of variables
- Similar to unfolding the computation
- Used daily on aeronautics software

Type systems

- A type is a set of (possible) values
- Checking
- Inference
- Polymorphism
- Non-standard type systems
  - view type system as a set of constraints to compute legal refactorings
  - use type inference to recover abstractions from optimized code

Model checking

- In simplest terms, exhaustive testing
  - Verify that every possible execution satisfies a given property
  - Very effective for hardware (inherently finite-state)
  - Popular for concurrent software
- How to make this scale?
  - Choose abstractions that lose just the right amount of precision
    - Counterexample-guided refinement
  - Efficient encodings

Analysis back-ends

- Reduce one problem to another
  - Often, produce a logical formula
- Reduction to SAT
  - 1979: "Problem *X* reduces to SAT, so it is hard."
  - 2009: "Problem *X* reduces to SAT, so it is easy."
- SMT (satisfiability modulo theories)
  - add non-logical constructs (e.g., arithmetic) to the logical formula
- Datalog (prolog-like; used in database community)
- Binary Decision Diagrams (BDDs)
- Boolean programs
- Theorem provers

Test generation

- Random
  - Scaleable, and more effective than you think
- Symbolic
  - What `if` statements guard a line of code?
  - Compute an input that satisfies them
- Concolic (**conc**rete + symb**olic**)
  - Run tests, then try to slightly modify them to achieve more coverage
- Evaluation of testing approaches

Dynamic analysis

- Testing
- Model creation
  - Observe executions, generalize from them
- Type inference
- Fault localization

Refactoring

- Refactoring changes program code without changing its meaning
- What constraints need to be generated to preserve the meaning?
- How to explore the space of solutions?

More

- Pointer and alias analysis
- Modeling and model-based development
- Configuration management
- Code generation and code completion
- Historical analysis
  - Prediction of bug-prone code

Applications

- Security
- Correctness
- Performance
- Rapid development
- System analysis
- Maintenance and evolution

Broader themes

- Precision vs. performance
- Power vs. transparency
- Static vs. dynamic
- Tuning analysis to the real problem

Format

- Lectures:
  - 50%: classic background
  - 50%: current research
  - Lectures are interactive (and, few slides)
- Homework:
  - Read research papers
  - 1 in-class presentation
- Group project to put the ideas into practice
  - Makes you a better researcher, in any field
  - You choose a topic (suggestions are provided)
  - Most projects lead to a publication or other research use
    - Not a requirement, just a common outcome

Who cares?

- Intellectually exciting and deep
- Spans both “hard” and “soft” areas of computing
- Connections to programming languages, security, systems, architecture, databases, and many more!
- Quals credit