Software Engineering Research

Premkumar Devanbu, Zhendong Su
Raju Pandey, Ron Olsson,
Hao Chen, Karl Levitt
What makes us tick?

**GOAL:** Produce software at lower cost, with fewer people, at a faster schedule.

**Approach:** Improve software engineering activities:

- Requirements
- Design
- Coding
- Quality Control
Example results.

- “I’ve invented a new language to program security in distributed systems that allows 3rd party development”
- “I’ve invented a new tool which automatically finds defects in programs that query databases”
- “I’ve discovered a way to predict defect rates in java classes based on their structure”
- “I’ve discovered a new way to organize software teams to produce IT applications faster, cheaper, and better”
- “I’ve discovered that what we believe about n-version programming is wrong, wrong, wrong”.
What’s the field like?

- Source of problems.
- Solutions come from related fields:
  - programming languages+compilers
  - algorithms
  - formal methods,
  - social science
What goes on here?

- Devanbu: programming models, middleware, software quality, open-source development.
- Su, Chen, Levitt: Software Quality, analysis, theory of programming languages.
- Pandey: programming models for new paradigms (sensor networks)
- Olsson: Concurrent Programming.
The Lay of the Land

- Major conferences: SIGSOFT, ICSE (~15% acceptance rate).
- Faculty Jobs: Usually more openings than candidates.
Main topics.

Improving Software Cost, Quality, Interval

- **Models**: theories, abstractions (e.g., UML, Z, Formal Logic, Petri nets)

- **Methods**: Procedures (e.g., Extreme Programming, coverage testing)

- **Tools**: Automation/Support (e.g., debuggers).
**Example: Model**

**Problem:** How do we develop distributed, heterogeneous systems?

**Solution:** Easier programming w/CORBA

**How?**

- Interface definition Language
- Tools to generate code
- Type-checked development.
- Run-time environment support

**Validation:**

Examples, Comparison with old way. Performance evaluation.
Problem: Developing concurrent systems is hard, e.g., device drivers

Solution: Find defects automatically in source code.

How?
* Abstract a finite-state model
* Describe the desired property
* Check the finite model.

Validation:
* Can we prove that it is sound?
* How efficient is it? Scaling?
* What is the rate of false positives?

Example: Tool
Example: Process

Problem: Allocating scarce inspection time.
Solution: Find defect-prone elements of systems.

How?
* Identify process goal & metric.
* Define plausible predictive product metrics
* Make statistical prediction.

Validation:
* Theoretically validate metrics (axiomatics).
* Statistical (non-parametric?) validation using historical data.
So, where is the field going?
What are the interesting problems?
How do I find a thesis topic?
How do I publish papers?
How do I find an academic job?
Burning Issues

- Separation vs. crosscutting
- Abstraction vs. Performance
- Protection vs. performance
- Agility, flexibility vs. Reliability, Quality
- Precision vs. Scaling
Separation vs. Crosscutting

**Goal:** Separation of concerns (why?)

**Problem:** Some concerns are hard to decompose (e.g., Security, Fault-tolerance, billing etc affect all components).

**Approaches:** Aspect-Oriented programming, Reflection, Monadic programming, Mixin Layers

**Issues:** Correctness, Efficiency, Understandability.
Abstraction vs. Performance

**Goal:** Brevity, Comprehensibility, SoC

**Problem:** Performance, and inflexibility.

**Approaches:** multi-layer optimization, partial evaluation.

**Issues:** Correctness, ease of use.
Protection vs. Performance

Goal: Protect critical resources

Problem: Inflexibility.

Approaches: “safe” extension mechanisms, such as sandboxes.

Issues: Correctness, power.
Agility and Flexibility vs. Reliability and quality.

**Goal:** The software process must be fast, flexible, and still be well controlled.

**Problem:** Control inhibits speed.

**Approaches:** Extreme Programming, Open source Development.

**Issues:** Applicability of these processes? Why do they work (specially open-source)?
Precision vs. Scalability.

Goal: Build analysis tools that find defects accurately.

Problem: Undecidability & combinatorial blow-up.

Approaches: Build sound but imprecise tools.

Succeeding in Research

- Read, read, read.
- Be a fashion plate and a name dropper.
- Attend Seminars: Systems, PL, but also security and theory.
- Talk, argue, canoodle, discuss.
- Question everything, and everyone.
Writing Papers-1

- The Role of Conferences.
- The reviewing process in conferences.
- The burden on the authors. Must write with extreme care!!!! Wordsmith!!
- Give your advisor a draft 2 weeks before the due date.
Writing Papers 2.

- Outline: Introduction, example(s), broad related work, solution, evaluation, narrow work, conclusion.
- Role of each section.
Writing Papers -3

- Introduction: Problem explained in broadest setting (clarify, don’t oversell).
- Example: Be current, simple, and to the point.
- Broad related work: Why is the example not solved?
Writing Papers-4

- Evaluation: consider the **culture** of your audience!!
  Formal, Examples, Performance studies, statistical validity etc.

- Close Related work: Be very precise, and non-judgement. Be shamelessly diplomatic. Look at the program committee, don’t be stupid.

- Conclusion. Summarize carefully, don’t oversell. Give web page for some software (even prototype).
Finding Academic Jobs

- Plan your graduation time carefully, based on your ambitions.
- Talk to your advisor about an external member.
- Go to workshops, conferences, chat up the big wigs. Ask them for letters.
- Have your advisor email colleagues in target universities.
Summary

- Exciting, inter-disciplinary field, requiring “lateral” thinking.
- The “action” is in managing tradeoffs of current interest.
- Conference papers are critical, and not easy.
- Academic job market is stable, and good.