The Gap Between Software Research and Practice

James Larus
Software Productivity Tools Group
Microsoft Research
August 2000
In theory, theory and practice are the same. In practice, they aren’t.

– anon
Past 10 Years

Major innovations in software development

- Object-oriented programming
  Simula/Smalltalk 60s-70s
- Safe GC languages
  Lisp/Algol/Pascal 50s-60s
- Components
  OLE/CORBA 80s-90s
- Design patterns
  Gang of 4 90s
Questions

- Why so few innovations?
- Why such a time lag?
- Why no new tools?
- Why so little research impact?

Answers based on observations at Microsoft
- Evolutionary development process
- Commercial software
Software Industry

- Software is huge, low productivity industry
  - $200B (?) cottage industry
- Software development is costly, frequently late, failure-prone
  - Product is low-quality
Software Development is Hard

- Produces most complex systems built by humans
  - Error intolerant
- Human as well as technical facets
  - Software built and used by fallible humans
  - Development is group activity
- Primitive languages and tools
  - Cave men chasing mammoths with spears?
Software Developers

- Smart and hardworking
- Aware of development shortcomings
  - Difficulty thinking outside the box
  - Suspicious of “solutions”
  - Intolerant of risk
- Microsoft developers above average
Critique of (Pure) Research

✔ Foibles
✔ Solutions
✔ Usability
Research Foibles

- Solutions in search of problems
- Quit after identifying difficulties
- Academic fads
Solutions not Papers

- Lot of hard work to go from good idea to usable tool
  - Jon Pincus’ s ISSTA talk (http://www.ics.uci.edu/IRUS/issta/)
- Entrepreneurs can bridge this gap
Something is Better than Nothing

- Publication favors small, neat results
  - Hopefully part of solution to large problem
- Reality forces incomplete, messy solutions
- Difficult to publish practical results
  - Negative result valuable too
Under-Appreciated Factors

- Scale
- Time to market
- Risk management
- Testing
Code Size

- Scale is critical issue in development
  - No one familiar with entire code
  - Continual change
- Small-scale ideas often don’t work
- Difficult to study scale in university
  - gcc is $2 \times 10^5$ LOC
  - Word97 is $2 \times 10^6$ LOC, Win2K is $3 \times 10^7$ LOC
Time to Market

- Time compression neglected in research
  - Not military/aviation 10 year procurements
  - 1-2yr release cycle (3-6mon on web)
  - Design time doesn’t compress

- Best is the enemy of the good
  - Can’t ship code if you are out of business
Development as Risk Management

- What can I risk doing?
  - Feature selection
  - Software reuse and sharing
  - Which bugs to fix

- Tools are risks
  - 50% chance of saving 1 month vs. 50% chance of wasting 1 month

- Tools and techniques have not demonstrated productivity improvement
Test, Test, and Test

- Only general technique for finding bugs and improving code quality
  - Huge amount of time and effort

- Academic orphan
  - Many incompleteness/NP-hardness results
  - Few new ideas
Discussion

- Software development poorly understood
  - No foundations
- Diverse research community working on point solutions
  - Little attention to most difficult problems
- Non-quantitative approach
  - What are benefits? How do you measure it?
Education

✓ No engineering orientation
✓ Weak skills
Educate Engineers, not Mechanics

- Development is/should be engineering
- Best students from best schools have skills of mechanics
  - Excellent programmers
- Not taught engineering mindset
  - Analyze problem, identify and apply best practices, and synthesize solution
Non-Solutions

- Software engineering major
  - Management ≠ engineering

- Resist vocational pressures
  - Teach how to write good programs, not to write Java programs
Basic Engineering Skills

- Details matter, so don’t cut corners
  - Does your problem set force a robust solution or just one that works?

- Reading and writing
  - Need to keep up with literature
  - Many cannot express ideas concisely or clearly

- Quantitative approach
  - Experimentation and measurement should not be foreign concept
Next 10 Years

- Easier to invent the future than to predict it.
  – Alan Kay
Next 10 Years

- **New & better tools**
  - Current ones are 30+ years old and text based
  - Semantic $\Rightarrow$ smart tools

- **Checkable, partial program specifications**
  - Provide clear benefits for extra work

- **Understanding & application of software architecture**
  - Packaged best practices

- **Software simulation**
  - Understand it before you build it
Path-Based Program Analysis

- Analyze individual paths through execution
  - Yes, there are too many, but most are unusable
  - Completeness not essential

- Precise error reports
  - e.g. Instrina’s Prefix, Compaq’s ECS/Java

- Software model checking
  - SLAM
Checkable, Partial Specs

- Programmers know far more than they say
  - Impoverished languages for describing behavior
  - Few tools exploit annotations $\Rightarrow$ little use

- Usable languages capture aspects of behavior
  - Develop hand-in-hand with analysis
  - e.g., Schneider’s security automata

- Vault language
  - Finite state properties embedded in type system
Software Architecture

- Package best practices
  - Design (patterns) & code (component) reuse

- Discover robust, scalable architectures
  - Case-by-case, domain specific
  - General theory?

- Analyze, package, present them
  - Impoverished language for talking about software

- Locality-Enhanced Staged Server (LESS)
Software Simulation

- Explore design space & test ideas without writing full system
  - Very hard to do in current languages
  - Write one to throw away

- Old idea: rapid prototyping
  - High-level languages produce concise, but inefficient executable specs
  - Machines fast & big enough
  - Component interfaces allow interoperability
Conclusion

- Academics should take first step across gap
  - Developers are too busy and skeptical
- Software development is rich source of research problems
- Better development is essential
  - Software is weak foundation of modern world
  - Development is expensive, error-prone bottleneck
Warnings!

- Academic perspective
  - 8 years on the faculty at Univ. Wisconsin
  - Run Software Productivity Tools (SPT) research group in Microsoft Research
- Not a software developer
- Focus on commercial, mass-market software
  - Applications, OS, web, …
  - Not safety-critical, military, or niche markets
- Not focused on key problems
- Academic results of little relevance
- Education does not teach engineering skills
Bridge the Gap

✓ Real-world experience
✓ Technology transfer
✓ Challenge problems
✓ Best practices
Get Your Hands Dirty

- Academics should better understand software development
  - Rich source of problems
  - Become better teachers

- Must get close to development
  - Start up a company, spend sabbatical in industry, visit research lab ...
Technology Transfer

- Publication is not enough
  - Developers read conference/journals rarely
- Reluctant to try published ideas
  - Most papers aren’t convince that idea works
- Risk in using something you don’t control
- Requires long-term relationship
Challenge Problems

- Beneficial to both parties
  - Force industry to identify prototypical problems
  - Focus research in relevant directions
- Basis for comparing results
- Easier to transfer results
- Motivation
  - eg Sequencing human genome
- SPT Device Driver Challenge
Best Practices

- Identify and record best practices
- Case studies of successful development
Undergraduate Education

- Teach undergraduates to think like engineers
  - May not have engineering underpinnings
  - Mindset is still useful
- Do not let “Software Engineering” define the curriculum
The Opportunity

- **Software development is rich source of research challenges**
  - Reduce development cost
  - Improve software quality

- **Better development is essential**
  - Software is rickety foundation for modern world