# Improving Undergraduate PL Curriculum

Kathleen Fisher
AT&T Labs Research

# ACM SIGPLAN
# Workshop on Programming Language Curriculum
# (PLC)

- Motivation: Initiate discourse on the role of programming languages in the undergraduate curriculum
- Co-chaired by Kathleen Fisher and Chandra Krintz
- Held May 29 & 30 at Harvard — Thanks!!
- Sponsored by NSF, NSA, and SIGPLAN — Thanks!!
- 30 participants
  - 16 steering committee members, 13 authors of selected white paper contributions, NSF and ACM Ed Board representatives

# Participants

Eric Allen (Sun Microsystems)
Mark Bailey (Hamilton College)
Ras Bodik (UC Berkeley)
Kim Bruce (Pomona College)
William Cook (UT Austin)
Matthias Felleisen (Northeastern Univ.)
Kathleen Fisher (AT&T Research)
Kathi Fisler (WPI)
Daniel Friedman (Indiana Univ.)
Stephen Freund (Williams College)
Sol Greenspan (NSF)
Robert Harper (CMU)
Michael Hind (IBM Research)
John Hughes (Chalmers)
Chandra Krintz (UC Santa Barbara)
Shriram Krishnamurthi (Brown)
Jim Larus (Microsoft Research)
Doug Lea (SUNY Oswego)
Gary Leavens (Univ. of Central Florida)
Greg Morrisett (Harvard Univ.)
Benjamin Pierce (Univ. of Pennsylvania)
Lori Pollock (Univ. of Delaware)
Stuart Reges (Univ. of Washington)
John Reynolds (CMU)
Martin Rinard (MIT)
Olin Shivers (Northeastern Univ.)
Peter Sestoft (ITU)
Mark Sheldon (Wellesley College)
Larry Snyder (Univ. of Washington)
Franklyn Turbak (Wellesley College)
Mitchell Wand (Northeastern Univ.)

# Mission Statement

- Explosive growth in CS in general and PL in particular
  - Internet, multi-core, managed runtime systems, etc.
- Most curricula have not kept pace
  - Some curricula no longer include a PL course at all
  - ACM/IEEE curriculum only minimally covers PL concepts
- Need to consider as a community

- **WHY** PL should be included in the CS curriculum
  - Clear articulation for non-PL academics of why every computer science undergraduate should have a solid PL knowledge base
- **WHAT** topics and concepts should be taught
  - Broad audience, many constraints, range of career paths and goals
  - To every student and to those choosing to study PL.
- **HOW** it should be taught
  - Recommended practices for range of venues, audiences, constraints

# Goals of the PLC Workshop

- Take ownership of the role of PL in our curricula.
- Provide a venue to initiate discussion on the Why, the What, and the How.
- Produce report to initiate community discussion, feedback, and on-going participation, containing:
  - Accepted white papers
  - Outcomes from workshop discussions
    - The Why
    - The What
    - And initial ideas on the How
  - To be published in SIGPLAN Notices November'08 issue
  - To be made available for community contribution
    - On the SIGPLAN webpage in September 2008

# Scope

- The focus of the workshop was on undergraduate PL curriculum.
- Given the limited time frame, we did not consider curricular questions concerning compilers, software engineering, and other related fields except as they directly relate to PL curriculum.
- Workshop participants felt these other areas should be considered as well in the future.

# The WHY: Students

- Address misconceptions & explore benefits of studying PL
- Misconceptions
  - Programming languages are boring.
  - Learning one language is all I need.
  - I can program in a language, so I know all I need to about PL.
  - I only care about one domain.
  - Imperative and object-oriented languages are the best/only models.
- Benefits
  - Learn new and precise ways of thinking computationally.
  - Use hot languages without getting burned (avoid pitfalls).
  - Recognize, conceptualize, and solve problems more productively.
  - Make your own language:  DSLs as a way of structuring code.
  - Increase job satisfaction and efficacy in the LONG term
    - Adapt easily to new domains and new programming languages.
    - Future languages will be built out of intellectual building blocks

# The WHY: Faculty

- Misconceptions
  - I never took PL and never needed it.
  - Computing advances all come from new algorithms and Moore's law.
  - PL is irrelevant: popular languages not designed by PL people.
  - No general principles; no intellectual depth.
  - Having multiple languages is bad -- we need one unifying one.
- Benefits
  - Many foundational concepts can be covered in a PL course.
    - Central to computer science and to core reasoning skills
  - Worldwide challenge: parallel and concurrent execution.
  - High impact: Web 2.0, map/reduce, code analysis.
  - Avoid software errors, security holes, performance bottlenecks
  - Provides new ways to reason about problems/identify solutions
  - Over career, students will use many different languages
    - Multiple languages commonly used in a single system

Myth: Students need to know only one language.

Reality: Students will use multiple languages (in both time and space)

- Over the course of a long career, students will use many different languages as new ones emerge.
- Most systems are multi-lingual, so developers need to use multiple languages even for a single system.
- Examples of broad range of languages used today:
  - General purpose programming languages
  - Regexps, CFGs, logics, ...
  - DSLs (map/reduce, contracts, bio)
  - XML, SGML, Scripting (outside vs. inside)
  - Video game control languages
  - Many models of computation: sequential, embedded,

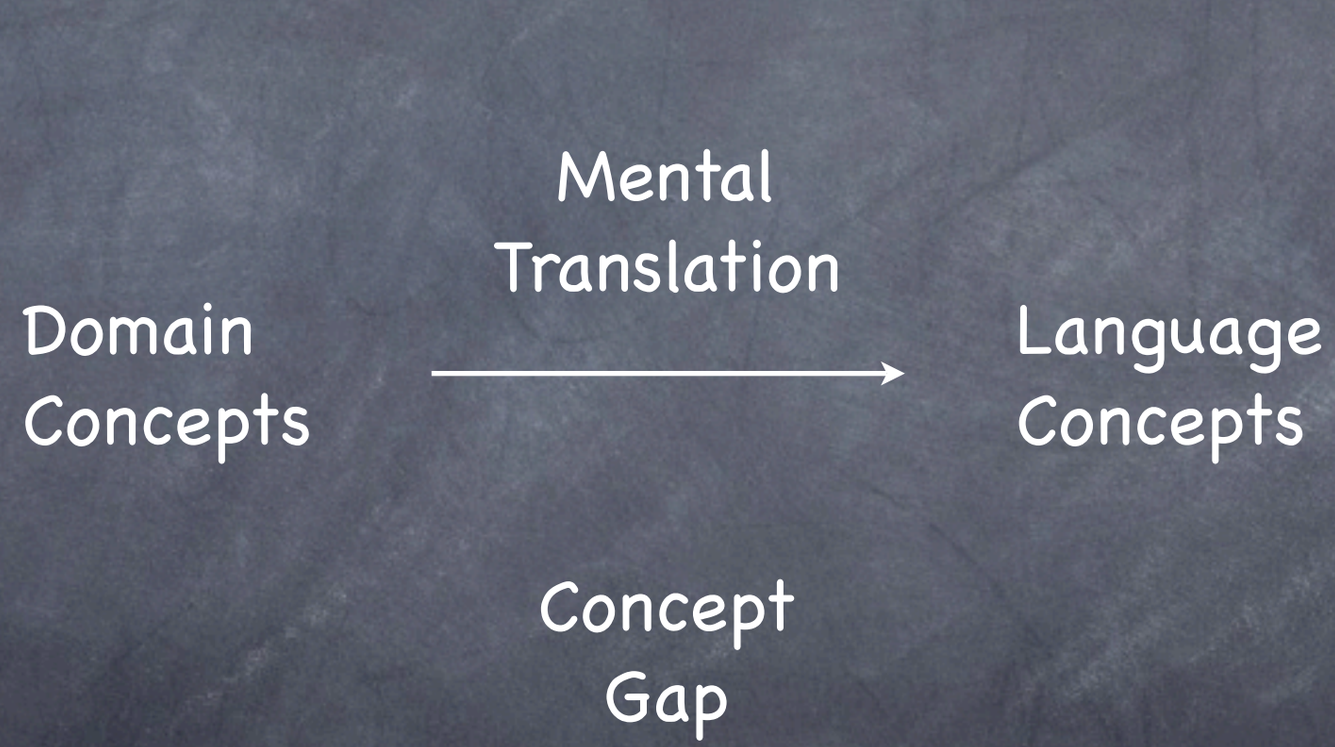
Myth: Almost no one ever designs a new language

Reality: Many people design small languages

- APIs, configuration files, XML schemas, are all embryonic languages
- Sometimes building a little language is the best way to solve a problem.

Myth: Having multiple languages is bad.
We need one unifying language.

Reality: Having multiple languages is beneficial, minimizes conceptual gap.



# Benefits of Minimizing Conceptual Gap

- More productive problem recognition, conception, & formulation.
- Fewer Errors
  - Can't express bad things
  - Can statically check for the absence of bad things.
- Minimizes cognitive burden on programmer;
  biases towards eliminating errors in solution.
- Preserves structure in solution
  - Supports easier analysis
  - Enables performance through compilation
  - Easier for domain experts to understand
  - Improves reliability
  - Facilitates maintenance

Myth: The important thing is the domain. There are no general principles in language design.

Reality: The domain is important, but general principles exist across domains.

- Compositionality
  - Abstraction, encapsulation, ...
  - Scope, naming, ...
  - Combinators, lambda, ...
  - Continuity principles: Locality of reasoning, translation
- Extensibility
  - procedures, types, modules, ...
- Precision
  - formal semantics, formal syntax
  - conceptual minimization: simple primitives, powerful combinators.
  - enabling for compilers, analysis tools, APIs, etc.

Myth: PL principles are irrelevant. Popular languages are not designed by programming language people.

Reality:

- Language designs depend on domain expertise.
- Many domain experts that design languages have had little exposure to language concepts.
  (Result: suboptimal language design)

Reality: Understanding principles enables people to:

- Design better embryonic and full languages
- Recognize limitations of existing languages and work around them
- Use concepts from advanced languages while working in more primitive languages

# General Principles are Important

- If don't follow principles, bad things happen.
  - Security vulnerabilities: SQL injection, cross-site scripting, ...
  - Errors because language is difficult to use effectively
  - Run into expressiveness and performance dead ends.
- If follow principles, good things happen.
  - Can specify precisely what programs will do
  - Minimizes surprises in use of constructs
  - Language generalizes for new problems in domain
  - Interacts better with other languages
  - Enables effective compilation
- Evidence: languages evolve to better conform to principles
  - TCL->Perl->Python->Ruby
  - C->C++->Java

Myth: The field of programming languages is just a hodgepodge of vocabulary and mechanisms with no intellectual depth.

Reality: Intellectually deep field

- Language principles
- Mathematical foundations
- Precision and development of general concepts
  - Abstraction, translation, automation
  - Expressive mechanisms (closures, continuations, ...)
  - Value of different/multiple perspectives

- Intellectual depth of field promotes general reasoning skills useful outside field.

# Opportunity: Parallelism

- May make programming more difficult so the language becomes more important.
- May need to preserve more domain structure in the program so need new/better languages.
- Opportunity to rethink software infrastructure because people will be open to new alternatives.

# Other arguments?

# The WHAT (All Undergrads)

All undergraduates should be able ANALYZE and APPLY

- Naming (binding, scope)
- Control (iteration, recursion, dyn. dispatch, exceptions, continuations?)
- Static/dynamic semantics
  - Invariants (loop, data structure)
  - Simple type systems, parametric polymorphism
  - Grammars (RE, CFG)
  - Static and dynamic typing
- Modularity and abstraction
  - Procedures
  - Compositionality, information hiding, classes
- Objects, state, mutation
- Higher-order functions, functional programming, immutability
- Runtime implementation (stacks, tail call, memory, GC)
  - Simple cost models (time/space complexity)
- Concurrency, parallelism
- Symbolic computation (programs as data)?

# The WHAT (PL Class)

- **UNDERSTAND, ANALYZE, and APPLY core concepts in PL**
  - Finite/infinite data structures, functions, control, concurrency, parallelism, state, modularity/interfaces, naming, cost models, laziness, monads
  - Models of computation
    - lambda calculus, FSAs, PDAs, relational calculus, Actors
  - Static and dynamic semantics
    - concrete/abstract syntax, type systems, transition systems, specifications

- **Know how to SYNTHESIZE into languages…**
  - OO, functional, logic/constraint programming, DSLs
- **And their USE in systems…**
  - Unix pipes, plan 9, TeX, nonces, modeling business processes, network protocols, OS schedulers, map/reduce, grep, web services, algorithmic analysis, tools to check systems

- **Know how to IMPLEMENT these concepts…**
  - interpreter, type checker, parser, translator, analysis tools

# The HOW

- Some ideas for reaching every student:
  - Improve PL content in CS1/CS2 intro courses
  - Add CS3: "Advanced Programming Techniques"
  - Integrate PL topics with other courses:
    Web Services (continuations, multiple languages), SE (modularity, specs), Computation (FSA, PDA, lambda), Language implementation (vms, compilers, interpreters), Systems (concurrency, naming, transactions)
  - Offer exciting PL Elective course that students want to take.

# The HOW: Improving CS1/CS2

Proposed "revenue neutral" change to required hours in the Computing Curriculum 2001 core:

http://www.sigcse.org/cc2001/cs-overview-bok.html#BOKTable

| Affected Knowledge Units (of 59 in PF/PL) | Current | Proposed |
|---|---|---|
| PF4 Recursion | 5 | 2 |
| PF5 Event-driven programming | 4 | 2 |
| PL1 Overview of PL | 2 | 0 |
| PL2 Virtual Machines | 1 | 0 |
| PL3 Language Translation | 2 | 0 |
| PL6 Object-oriented programming | 10 | 10 |
| PL7 Functional Programming | 0 | 10 |
| Total Number of Hours | 24 | 24 |

# ACM Curriculum Review

- Accepted proposal into draft revision
- Posted all revisions to web site for comment:
  - http://wiki.acm.org/cs2001/index.php
  - Need ACM account/password to comment.
  - Open for comments June 9th to July 1st
  - As of 6/15/08, 99 comments, 97 highly positive.
  - Add your comments!
- ACM Committee will respond to all comments and finalize recommendation.
- Full curriculum review will start shortly thereafter.

# An Opportunity...

- If curriculum is adopted, many schools will need 10 hours of functional programming instruction and not know how to do it.
- The functional programming community knows how to do it and has lots of instructional material.
- If we build "packages" of 10 hours of material and collect them on a high-profile site, we have a chance to get high quality functional programing instruction in place in many institutions.

# Broader Strategies

- Develop clear and convincing WHY materials.
- Develop and document PL community consensus on WHAT/HOW
- Work to influence the ACM/IEEE curriculum
- Constitute a SIGPLAN Education board
  - Members interact with ACM/IEEE Ed boards
  - Solicit community white papers on why/what/how
  - Monitor and moderate improvements to why/what/how
  - Highlight good curricula, course materials, and textbooks.

# Call for Feedback & Participation

- **To impact curricula, we need broad community involvement**
- How to get involved
  - Provide feedback on proposed changes to ACM curriculum.
  - Contribute materials for "10 hours of FP."
  - Help make the Why case.
  - Provide feedback on the PLC Workshop Report, to be published in November SIGPLAN Notices.
  - Contribute to SIGPLAN Educational Board (forming soon)
  - Inform and involve others
  - Help organize WPLC in '09 and beyond
  - **Contact Kathleen or Chandra** with feedback or to volunteer **{chair_sigplan,vc_sigplan}@acm.org**