

# PDC Modules for Every Level:

*A Comprehensive Model for  
Incorporating PDC Topics into the  
Existing Undergraduate Curriculum*

EduPar Presentation · May 2011 · Anchorage, AK, USA

Konstantin Läufer (presenter)  
Chandra Sekharan (dept. chair)  
George K. Thiruvathukal  
*Loyola University Chicago*

# Presentation Outline

- Institutional profile
- The past: our early single-course approach
- The present: our later cross-curricular approach
- The near future: proposed required core modules
- The future: proposed advanced/elective modules
- Tying everything together
- Position statement

# Institutional Profile

- Loyola U. Chicago: urban, private, Jesuit, liberal arts, ~16k
  - College of Arts and Sciences, ~8k
    - Department of Computer Science, ~200
- 9 full-time faculty
  - 8 CS (7 TTT, 1 clinical)
  - 1 bioinformaticist (1/2 FTE)
  - 1 algebraist (1/2 FTE)
- 100+ undergrad majors in CS, SE, IT, Networks/Security
- 80+ master's students in CS, SE, IT
- External funding: NSF S-STEM, NSF BPC lead institution, NSF research grants, industry grants and donations

# Where Our Graduates Go...

- Industry
  - midwest, coasts, international
  - consulting, finance, software, telecom, ...
- Academia and Government
  - Argonne, county admin, local universities
- Graduate School
  - local, national
- Professional Schools
  - business, law, medical

# Where Our Graduates Go...

- Industry **80%\***
  - midwest, coasts, international
  - consulting, finance, software, telecom, ...
- Academia and Government **15%\***
  - Argonne, county admin, local universities
- Graduate School **3%\***
  - local, national
- Professional Schools **2%\***
  - business, law, medical

\*guesstimates

# ...and What They Need to Know

Most of them need to know

- Parallel and Distributed Computing
  - especially programming and algorithm topics

# ...and What They Need to Know

Most of them need to know *both*\*

- Parallel and Distributed Computing
  - especially programming and algorithm topics
- Software Engineering\*
  - methodology/process
  - software architecture & design patterns
  - languages and tools
  - collaboration/social coding/FOSS

\* will explore this thought later

# We Are Very Early PDC Adopters

- We have been teaching our students explicit PDC topics since spring 1997.



# We Are Very Early PDC Adopters

- We have been teaching our students explicit PDC topics since spring 1997.
- Active research program in relevant areas
- NSF research grants
- Industry grants and donations
- Thiruvathukal's book *High Perf. Java Platform Computing* (lives on at [hpjpc.googlecode.com](http://hpjpc.googlecode.com))
- Paper in OOPSLA 1998 Educator Symposium

# We Are Very Early PDC Adopters

- We have been teaching our students explicit PDC topics since spring 1997.
- Active research program in relevant areas
- NSF research grants
- Industry grants and donations
- Thiruvathukal's book *High Perf. Java Platform Computing* (lives on at [hpjpc.googlecode.com](http://hpjpc.googlecode.com))
- Paper in OOPSLA 1998 Educators' Symposium
- **We are eager supporters of the NSF/IEEE-TCPP Curriculum Initiative!**

# The Past: Our Early Single-Course Approach

- Level: 2nd or 3rd year
- Prerequisite: Intermediate Object-Oriented Development
- Text: Doug Lea, *Concurrent Programming in Java*, Addison-Wesley, 1997.
- PDC: thread- and event-based concurrent programming
  - paradigms
  - notions
  - semantics and correctness issues
- Additional perspectives:
  - software design patterns, e.g., Observer
  - software architecture, e.g., layering
  - automated testing

# The Past: Our Early Single-Course Approach [OOPSLA 98 Edu Symposium]

- Level: 2nd or 3rd year
- Prerequisite: Intermediate Object-Oriented Development
- Text: Doug Lea, *Concurrent Programming in Java*, Addison-Wesley, 1997.
- PDC: thread- and event-based concurrent programming
  - paradigms
  - notions
  - semantics and correctness issues
- Additional perspectives:
  - software design patterns, e.g., Observer
  - software architecture, e.g., layering
  - automated testing
- Anecdotal evidence of success: feedback from students and employers

# The Present: Our Later Cross-Curricular Approach

- In response to departmental staffing and scheduling needs
- *Fuzzy learning units* for different PDC topics
- Incorporated in advanced/elective courses (at least two offered per semester)
  - CS 322: Software Development for Wireless/Mobile Devices
  - CS 338: Server-Side Software Development
  - CS 339: Distributed Systems
  - CS 342: Web Services Programming
  - CS 364: High-Performance Computing
  - CS 372: Programming Languages (*Lang*)
  - CS 373: Advanced Object-Oriented Development

# The Present: Our Later Cross-Curricular Approach

- In response to departmental staffing and scheduling needs
- *Fuzzy learning units* for different PDC topics
- Incorporated in advanced/elective courses (at least two offered per semester)
  - CS 322: Software Development for Wireless/Mobile Devices
  - CS 338: Server-Side Software Development
  - CS 339: Distributed Systems
  - CS 342: Web Services Programming
  - CS 364: High-Performance Computing
  - CS 372: Programming Languages (*Lang*)
  - CS 373: Advanced Object-Oriented Development
- Exposure varies widely across students and semesters

# The Near Future: *Our Proposed Set of Required Core Modules*

Goal: regularly and consistently expose all undergraduate majors to PDC core knowledge

# The Near Future: Our Proposed Set of Required Core Modules

Goal: regularly and consistently expose all undergraduate majors to PDC core knowledge

Approach:

- push down into *required existing* 2nd-year foundation courses

identify suitable topics from *TCPP 45h sample course* (mostly “K” and “C” level, some “A”)

- package as three-week core PDC modules (20% of our 15-week semester or 30% of a 10-week quarter = 9 hours) → **36h total**



# Common Undergraduate Foundation

- Calculus I
- CS0 + CS1 + CS2 (*Core*)
- Discrete Structures (*Core, DM*)
- CS 264: Intro to Computer Systems (*Core, Systems*)
- CS 313: Intermediate Object-Oriented Dev (CS & SE only)
- Intro to Scientific and Technical Communication
- Social, Legal, and Ethical Issues in Computing
- Practicum (6 credits, in-house or external)

# Common Undergraduate Foundation

- Calculus I
- CS0 + CS1 + CS2 (*Core*)
- Discrete Structures (*Core, DM*)
- CS 264: Intro to Computer Systems (*Core, Systems*)
- CS 313: Intermediate Object-Oriented Dev (CS & SE only)
- Intro to Scientific and Technical Communication
- Social, Legal, and Ethical Issues in Computing
- Practicum (6 credits, in-house or external)

## Other Relevant Existing Courses

- CS 363: Design & Analysis of Comp Alg (*Core, DS/A*)
- CS 372: Programming Languages (*Advanced, Lang*)
- CS 330: Software Engineering (*Advanced, SwEngg*)

# PDC Core Module: *Introduction to PDC*

- every semester
- in CS2
- intro topics (arch, prog, algo, “K” and “C” level)
  - target machine models (1.5h)
  - parallel control statements (1.5h)
  - shared memory language extensions & libraries (1.5h)
  - tasks, threads, and synchronization (3h)
  - searching and sorting (1.5h)
- C# as the teaching language (at least for this module)
  - well-designed mechanisms that support these topics
  - foundationally sound teaching materials [Ball et al.]
  - cross-platform via Mono Project

# PDC Core Module: *Architecture*

- every fall
- in CS 264 (Systems)
- architecture topics
  - high-level themes (1.5h)
  - classes (4.5h)
    - taxonomy
    - data versus control parallelism
    - shared versus distributed memory
  - memory hierarchy, caches (1h)
  - floating-point representation (0.5h)
  - performance metrics (1h)
  - power Issues (0.5h)

# PDC Core Module: *Programming*

- every semester (CS & SE majors)
- in CS 313 (intermediate object-oriented development)
- programming topics (*some up to “A” level*)
  - selected parallel programming notations (1.5h)
  - semantics and correctness issues (4.5h)
    - tasks and threads
    - synchronization
    - defects
  - performance issues (1.5h)
  - tools (1.5h)
- C# as the teaching language for the entire course
  - threads, actors, tasks
  - events
  - software transactional memory

# PDC Core Module: *Algorithms*

- every spring (CS majors)
- in CS 363 (Algo)
- algorithm topics
  - parallel/distributed models and complexity (4h)
    - cost of computation, scalability: asymptotics, time, cost, work, speedup, efficiency, space, power
  - algorithmic paradigms (3h)
    - divide and conquer, recursion
    - series-parallel composition
  - algorithmic problems (2h)
    - synchronization
    - specialized computations

# The Future: *Our Proposed Set of Advanced/Elective Modules*

- slated for development after core modules
- each module typically offered every three semesters
- in suitable electives (from list on slide “*The Present*”)
- **Advanced Programming:** parallel prog. and concurrency topics from PL principles and paradigms perspective, using F# or Scala for programming projects
- **Distributed Foundations:** foundational topics including architecture classes, models and complexity, and concurrency topics
- **Distributed Programming and Applications:** languages, frameworks, and software architectures for distributed computing, semantics and correctness issues, performance issues, and advanced topics

# Tying Everything Together: Roadmap

- summer 2011: develop the core PDC modules
- fall 2011: start offering core PDC modules
- starting summer 2011: develop PDC modules for advanced/elective courses



# Tying Everything Together: Evaluation and Dissemination Plans

- key aspect of this proposal
- qualitative and quantitative measurement
- longitudinal measurement over three to five years
- refine our evaluation plan further by working with
  - TCPP
  - fellow early adopters
  - Loyola's Center for Science & Math Education
- hold workshops for subsequent adopters in the Midwest

# Position Statement

[for further discussion]

To teach PDC topics effectively, they should not be taught in isolation. Instead, they should be taught in conjunction with relevant software engineering best practices.

# Position Statement

[for further discussion]

To teach PDC topics effectively, they should not be taught in isolation. Instead, they should be taught in conjunction with relevant software engineering best practices.

## Examples

- methodology/process
- software architecture
- software design patterns
- automated testing
- continuous integration
- collaboration/social coding/FOSS
- languages: object-oriented, functional, scripting, parallel
- tools: IDE, (D)VCS, build manager, doc generator

# Position Statement

[for further discussion]

To teach PDC topics effectively, they should not be taught in isolation. Instead, they should be taught in conjunction with relevant software engineering best practices.

Examples ← “How do you know?”

- methodology/process
- software architecture
- software design patterns
- automated testing
- continuous integration
- collaboration/social coding/FOSS
- languages: object-oriented, functional, scripting, parallel
- tools: IDE, (D)VCS, build manager, doc generator

Talk to the Practitioners!

# E.g., ThoughtWorks Technology Radar

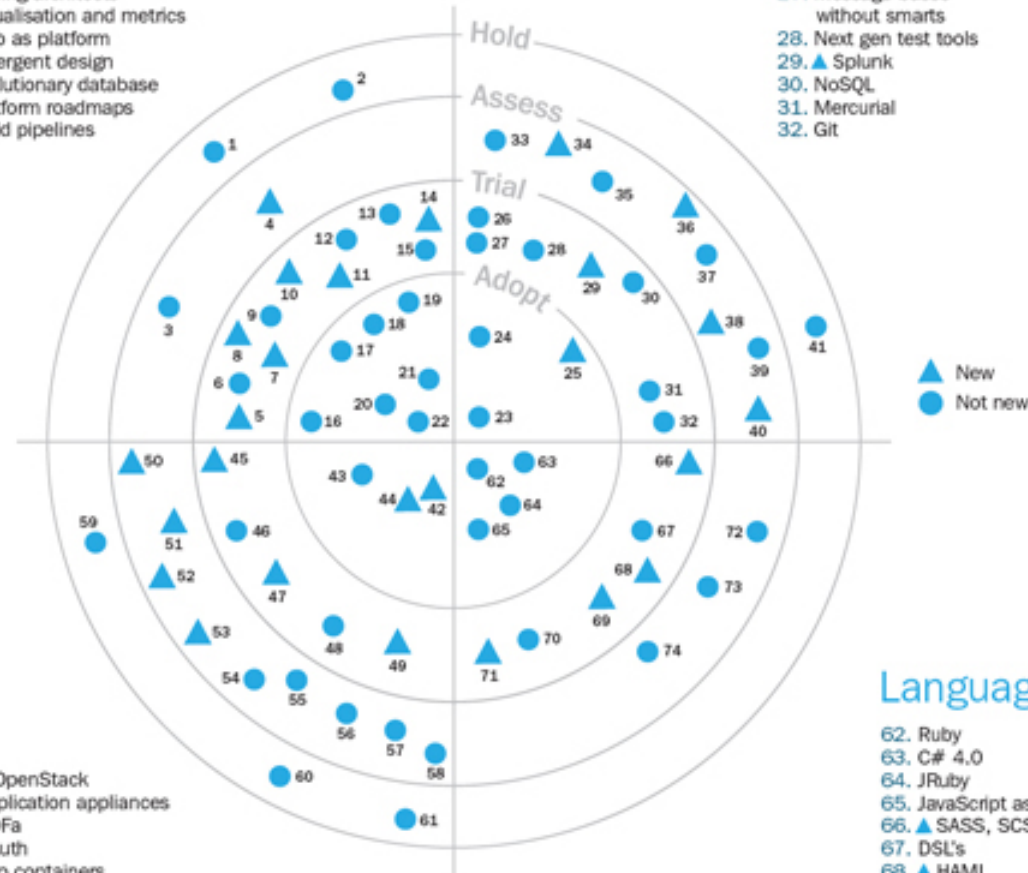
## Technology Radar

### Techniques

1. Database based integration
2. Scrum certification
3. Real-time business intelligence
4. ▲ Smart Systems
5. ▲ Progressive Enhancement
6. Automation of technical tests
7. ▲ Automate database deployment
8. ▲ Concurrency abstractions and patterns
9. Capability modelling
10. ▲ Acceptance test of journeys
11. ▲ DevOps
12. Service choreography
13. Continuous deployment
14. ▲ Categorisation & prioritisation of technical debt
15. Evolutionary architecture
16. Coding architects
17. Visualisation and metrics
18. Web as platform
19. Emergent design
20. Evolutionary database
21. Platform roadmaps
22. Build pipelines

### Tools

23. Subversion
24. Squid
25. ▲ Infrastructure as code
26. Apache camel
27. Message buses without smarts
28. Next gen test tools
29. ▲ Splunk
30. NoSQL
31. Mercurial
32. Git
33. Cross mobile platforms
34. ▲ Deltacloud
35. Github
36. ▲ Vagrant
37. Restfulie
38. ▲ WCF HTTP
39. RDF triple stores
40. ▲ API management services
41. ESB



### Platforms

42. ▲ KVM
43. Android
44. ▲ Atom
45. ▲ Heroku
46. Facebook as business platform
47. ▲ iPad
48. EC2 & S3
49. ▲ Mobile Web
50. ▲ GPGPU
51. ▲ Node.js
52. ▲ vFabric
53. ▲ OpenStack
54. Application appliances
55. RDFa
56. OAuth
57. App containers
58. Azure
59. WS-\* beyond basic profile
60. GWT
61. RIA

### Languages

62. Ruby
63. C# 4.0
64. JRuby
65. JavaScript as a first class language
66. ▲ SASS, SCSS, and LESS
67. DSL's
68. ▲ HAML
69. ▲ Scala
70. Groovy
71. ▲ HTML 5
72. Java language end of life
73. F#
74. Clojure