Empirical Software Engineering Research at UMD

Victor R. Basili

University of Maryland
and
Fraunhofer Center - Maryland
Setting the Context

- Software engineering is an engineering discipline
- We need to understand products, processes, and the relationship between them (*we assume there is one*)
- We need to experiment (human-based studies), analyze, and synthesize that knowledge
- We need to package (model) that knowledge for use and evolution
- Recognizing these needs changes how we think, what we do, what is important
For example, a software organization needs to ask:
What is the right combination of technical and managerial solutions?
What are the right set of process for that business?
How are they tailored?
How do they learn from their successes and failures?
How do they demonstrate sustained, measurable improvement?

More specifically:
When are peer reviews more effective than functional testing?
When is an agile method appropriate?
When do I buy rather than make my software product elements?
Examples of Useful Empirical Results

“Under specified conditions, …”

Technique Selection Guidance

- **Peer reviews** are more effective than functional testing for faults of omission and incorrect specification (UMD, USC)
- **Functional testing** is more effective than reviews for faults concerning numerical approximations and control flow (UMD, USC)

Technique Definition Guidance

- For a reviewer with an average experience level, a **procedural approach** to defect detection is more effective than a less procedural one. (UMD)
- Procedural inspections, based upon **specific goals**, will find defects related to those goals, so inspections can be customized. (UMD)
- Readers of a software artifact are more effective in uncovering defects when each uses a **different and specific focus**. (UMD)
Basic Concepts for Empirical Software Engineering

The following concepts have been applied in a number of organizations

**Quality Improvement Paradigm (QIP)**

An evolutionary learning paradigm tailored for the software business

**Goal/Question/Metric Paradigm (GQM)**

An approach for establishing project and corporate goals and a mechanism for measuring against those goals

**Experience Factory (EF)**

An organizational approach for building software competencies and supplying them to projects
The Experience Factory Organization

Project Organization

1. Characterize
2. Set Goals
3. Choose Process

Execution plans

4. Execute Process

Experience Factory

environment characteristics
tailorable knowledge, consulting
products, lessons learned, models
project analysis, process modification
data, lessons learned

Project Support

Experience Base

5. Analyze

6. Package
Generalize
Tailor
Formalize
Disseminate
## The Experience Factory Organization

### A Different Paradigm

<table>
<thead>
<tr>
<th>Project Organization Problem Solving</th>
<th>Experience Factory Experience Packaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition of a problem into simpler ones</td>
<td>Unification of different solutions and re-definition of the problem</td>
</tr>
<tr>
<td>Instantiation</td>
<td>Generalization, Formalization</td>
</tr>
<tr>
<td>Design/Implementation process</td>
<td>Analysis/Synthesis process</td>
</tr>
<tr>
<td>Validation and Verification</td>
<td>Experimentation</td>
</tr>
<tr>
<td><strong>Product Delivery within Schedule and Cost</strong></td>
<td><strong>Experience / Recommendations Delivery to Project</strong></td>
</tr>
</tbody>
</table>
CeBASE Project Goal: Enable a decision framework and experience base that forms a basis and infrastructure needed to evaluate and choose among software development technologies

CeBASE Research Goal: Create and evolve an empirical research engine for building the research methods that can provide the empirical evidence of what works and when

Partners: Victor Basili (UMD), Barry Boehm (USC)
CeBASE Approach

Observation and Evaluation Studies of Development Technologies and Techniques → Empirical Data

Predictive Models (Quantitative Guidance)

General Heuristics (Qualitative Guidance)

E.g. COCOTS excerpt:

Cost of COTS tailoring = f(# parameters initialized, complexity of script writing, security/access requirements, …)

E.g. Defect Reduction Heuristic:

For faults of omission and incorrect specification, peer reviews are more effective than functional testing.
CeBASE
Three-Tiered Empirical Research Strategy

Technology maturity

- Practical applications
  - Government, industry, academia

- Applied Research

- Basic Research

Primary activities

- Practitioner use, tailoring, and feedback. Maturing the decision support process.

- Experimentation and analysis with the concepts in selected areas.

- Building a SE Empirical Research Engine and Experience base structure

Evolving results

- Increasing success rates in developing agile, dependable, scalable applications.

- Partly filled EB, more mature empirical methods, technology maturation and transition.

- Empirical methods for SE, Experience Base definition, decision support structure
CeBASE Basic Research Activities

Define and improve methods to

• Formulate evolving hypotheses regarding software development decisions

• Collect empirical data and experiences

• Record influencing variables

• Build models (Lessons learned, heuristics/patterns, decision support frameworks, quantitative models and tools)

• Integrate models into a framework

• Testing hypotheses by application

• Package what has been learned so far so it can be evolved
Applied Research
NASA High Dependability Computing Program

**Problem:** How do you elicit the software dependability needs of various stakeholders and what technologies should be applied to achieve that level of dependability?

**Project Goal:** Increase the ability of NASA to engineer highly dependable software systems via the development of new technologies in systems like Mars Science Laboratory

**Research Goal:** Quantitatively define dependability, develop high dependability technologies and assess their effectiveness under varying conditions and transfer them into practice

**Partners:** NASA, CMU, MIT, UMD, USC, U. Washington, Fraunhofer-MD
What are the top level research problems?

**Research Problem 1**
Can the quality needs be understood and modeled?

**Research Problem 2**
What does a technology do?
Can it be empirically demonstrated?

**Research Problem 3**
What set of technologies should be applied to achieve the desired quality? (Decision Support)
System User Issues

How do I elicit quality requirements?
How do I express them in a consistent, compatible way?

• How do I identify the non-functional requirements in a consistent way?
  – Across multiple stakeholders
  – In a common terminology (Failure focused)
  – Able to be integrated
• How can I take advantage of previous knowledge about failures relative to system functions, models and measures, reactions to failures?
  – Build an experience base
• How do I identify incompatibilities in my non-functional requirements for this particular project?
The **Unified Model of Dependability** is a requirements engineering framework for eliciting and modeling quality requirements.

Requirements are expressed by specifying the actual *issue* (failure and/or hazard), or class of issues, that should not affect the system or a specific service (*scope*).

As issues can happen, tolerable manifestations (*measure*) may be specified with a desired corresponding system *reaction*. External *events* that could be harmful for the system may also be specified.

For an on-line bookstore system, an example requirement is:

“The book search service (scope) should not have a response time greater than 10 seconds (issue) more often than 1% of the cases (measure); if the failure occurs, the system should warn the user and recover full service in one hour”.
UMD is a model builder

**Scope**
- **Type**
  - Whole System
  - Service
- **Operational Profile**
  - Distribution of transaction
  - Workload volumes
  - etc.

**Measure**
- **Measurement Model**
  - MTBF
  - Probability of Occurrence
  - % cases
  - MAX cases in interval X
  - Ordinal scale
    (rarely/sometimes/...)

**Issue**
- **Failure**
  - **Type**
    - Accuracy
    - Response Time
    - etc.
  - **Availability Impact**
    - Stopping
    - Non-Stopping
  - **Severity**
    - High
    - Low
- **Hazard**
  - **Severity**
    - People affected
    - Property only
    - etc.

**Event**
- **Type**
  - Adverse Condition
  - Attack
  - etc.

**Reaction**
- **Impact mitigation**
  - warnings
  - alternative services
  - mitigation services
- **Recovery**
  - recovery time / actions
- **Occurrence reduction**
  - guard services

**Concern**

**Cause**

**Manifest**

**Trigger**
Characterizations (e.g., types, severity, etc.) of the basic UMD modeling concepts of issue, scope, measure, and event depend on the specific context (project and stakeholders).

They can be customized while applying UMD to build a quality model of a specific system and enriched with each new application.
Technology Developer Issues

How well does my technology work? Where can it be improved?

• How do I articulate the goals of a technology?
  – Formulating measurable hypotheses

• How do I empirically demonstrate its goals?
  – Performing empirical studies
  – Validate expectations/hypotheses

• What are the requirements for a testbed?
  – Fault seeding

• How do I provide feedback for improving the technology?
Example Technology Evolution

A process for inspections of Object-Oriented designs was developed using multiple iterations through this method.

Early iterations concentrated on feasibility:
- effort required, results due to the process in the context of offline, toy systems.

Is further effort justified?

Mid-process iterations concentrated on usability:
- usability problems, results due to individual steps in the context of small systems in actual development.

What is the best ordering and streamlining of process steps to match user expectations?

Most recent iterations concentrated on effectiveness:
- effectiveness compared to other inspection techniques previously used by developers in the context of real systems under development. Does the new techniques represent a usable improvement to practice?
Using testbeds to transfer technology

• A **testbed** is a set of artifacts and the infrastructure needed for running experiments, e.g., evaluation support capabilities such as instrumentation, seeded defect base; experimentation guidelines, specific features to monitor faults, …

• **Used to**
  – Conduct empirical evaluations of emerging technology
  – Stress the technology and demonstrate its context of effectiveness
  – Help the researcher identify the strengths, bounds, and limits of the particular technology at different levels
  – Provide insight into the integration of technologies
  – Reduce costs by reusing software artifacts
  – Reduce risks by enabling technologies to mature
  – Assist technology transfer of mature results
Example Technology and Testbed Evolution

- **Testbed**: a safety critical air traffic control software component (FC-MD’s TSAFE III)

- **Technology**: Tevfik Bultan’s model checking design for verification approach applied to concurrent programming in Java

- **Technology goal**: Eliminate synchronization errors techniques

- **Empirical Study Goal**: investigate the effectiveness of the design for verification approach on safety critical air traffic control software
  - Applied the design for verification approach to a safety critical air traffic control software component (FC-MD’s TSAFE III)
  - TSAFE III software was reengineered based on the concurrency controller design pattern
Example Technology and Testbed Evolution

- **Testbed:**
  - 40 versions of TSAFE source code were created via fault seeding
  - The faults were created to resemble possible errors that can arise in using the concurrency controller pattern such as:
    • making an error while writing a guarded command or
    • forgetting to call a concurrency controller method before accessing a shared object

- **Results:**
  - The experimental study resulted in a
    • Better fault classification
    • Identified strengths and weaknesses of the technology
    • Helped improve the design for verification approach
  - However, there was one type of fault that was difficult to catch
    • Three uncaught faults were created to test this
System Developer Issues

How can I understand the stakeholders dependability needs?
How can I apply the available techniques to deliver the required dependability?

- How do I identify what dependability properties are desired?
  - Stakeholders needs, dependability goals and models, project evaluation criteria
- How do I evaluate the effectiveness of various technologies for my project?
  - What is the context for the empirical studies?
- How do I identify the appropriate combinations of technologies for the project needs?
  - Technologies available, characterization, combinations of technologies to achieve goals
- How do I tailor the technologies for the project?
Problem: How do you improve the time and cost of developing high end computing (HEC) codes?

Project Goal: Improve the buyers ability to select the high end computer for the problems to be solved based upon productivity, where productivity means

Time to Solution = Development Time + Execution Time

Research Goal: Develop theories, hypotheses, and guidelines that allow us to characterize, evaluate, predict and improve how an HPC environment (hardware, software, human) affects the development of high end computing codes.

Partners: MIT Lincoln Labs, MIT, UCSD, UCSB, UMD, USC, FC-MD
Motivation

• Important types of programs cannot be run on desktop PCs – e.g. many simulations of physical phenomena like:
  – Climate modeling
  – Computational fluid dynamics
  – Protein folding
  – Nuclear bomb simulation
  – Crash simulation
• They require high-performance computing (HPC) systems
  – a.k.a. high-end computing, supercomputing, parallel computing

Many studies of performance of algorithms on HPC machines

...BUT...

Very few studies of productivity of programmers on HPC machines
Motivation

- US Govt. wants to buy new HPC systems for national labs
- Criteria for choosing machines: minimize time to solution (TTS)

\[ TTS = Development \ time + Computing \ time \]

- Not all labs are the same
  - Programmers may have different levels of experience
    - One system may be more productive for experts, another for novices
  - Problems being solved are different
    - One system may be better for climate modeling, another for protein folding

- Research question:
  - Which systems and development approaches will result in higher productivity for given users, problem domains, etc.?
Overall Research Plan

HPC community provides questions to study leading to successively larger and more complex experiments.

- Single programmer (expert studies)
  - Kernels
  - Class assignments
- Single programmer classroom Studies and Observational studies
- Team projects
- Class projects
- Case studies
  - porting
  - compact applications

Program Duration
Evolving measurements, Models, Hypotheses
Why not apply “traditional” lessons of software development productivity?

Critical aspects of HPC programming are very different:

- **Programmers**: physical scientists (not computer scientists)
  - Not expert programmers
- **HPC architectures**: Problem and development approach must be compatible with machine architecture
  - No compilers exist that can shield programmers from architecture details
  - Few tools support HPC programming
- **Performance tuning**: Significant effort spent optimizing performance across processors
- **Quality assurance**: Programs usually written to expand scientific understanding
  - Often do not know the correct output in advance: difficult to test!
We will use these experiments to build a knowledge base of results.

Experimental designs
Hypotheses

Folklore/Results

Single Programmer Applications

Heroic Scale Applications

Blue Collar Applications

KNOWLEDGE BASE
-Quantitative insights
-Models in context

Folklore/Results

Insights, Models, Results

scale
HPCS Example Questions

- How does a HEC environment (hardware, software, human) affect the development of an HEC program?
  - What is the **cost** and **benefit** of applying a particular HPC technology (MPI, Open MP, UPC, Co-Array Fortran, XMTC, StarP,...)?
  - What are the **relationships** among the technologies, the work flows, development cost, the defects, and the performance?
  - What **context variables** affect the development cost and effectiveness of the technology in achieving its product goals?
  - Can we build **predictive models** of the above relationships?
  - What **tradeoffs** are possible?
  - ...
HPCS Research Activities

Development Time Experiments – Novices and Experts → Empirical Data 

Predictive Models (Quantitative Guidance)

E.g. Tradeoff between effort and performance:

MPI will increase the development effort by y% and increase the performance z% over OpenMP

General Heuristics (Qualitative Guidance)

E.g. Experience:

Novices can achieve speed-up in cases X, Y, and Z, but not in cases A, B, C.
HPCS Testbeds

We are experimenting with a series of testbeds ranging in size from:

- **Classroom assignments** (Array Compaction, the Game of Life, Parallel Sorting, LU Decomposition, …)

  to

- **Compact Applications** (Combinations of Kernels, e.g., Embarrassingly Parallel, Coherence, Broadcast, Nearest Neighbor, Reduction)

  to

- **Full scientific applications** (nuclear simulation, climate modeling, protein folding, ….)
Independent Variable: Problem Type

Different problems can be decomposed differently:

- Embarrassingly parallel
  - No communication necessary
  - Example: Monte Carlo simulations
- Nearest neighbor
  - Communication among “neighbors”
  - Example: “Game of Life”
- All-to-all
  - All processes communicate with each other
  - Example: FFT

- Generally, goal is to maximize computation / communication ratio
Design Implications

• Independent variables:
  – Problem type
  – Parallel programming model
    • MPI, OpenMP, Co-Array Fortran, UPC, Matlab Star-P, XMT-C
  – Developer experience

• Dependent variables:
  – Speedup
  – Effort
    • Manual logs
    • Program snapshots at every compile (sometimes every keystroke)
  – Code expansion factor
  – Cost per LOC
Data collection software

Experimental artifacts

Programming problems

Experimental Packages

Classroom studies

Industrial studies

Data collection software

Advice to mission partners
- Workflow models
- Productivity models

Advice to university professors
- Effective programming methods
- Student workflows

Advice to vendors
- Language features utilization
- Workflow models
Studies Conducted

- UCSB: 3 studies
- USC: 4 studies
- UCSD: 1 study
- UCSD: ASCI Alliance
- U Utah ASCI Alliance
- UIUC ASCI Alliance
- U Chicago ASCI Alliance
- MIT: 3 studies
- UMD: 6 studies
- Mississippi State: 2 studies
- Iowa State: 1 study
- Stanford U: ASCI Alliance
- CalTech: ASCI Alliance
- UIUC: ASCI Alliance
- U Utah: ASCI Alliance
- U Chicago: ASCI Alliance
- U Utah: ASCI Alliance
Motivation for Empirical Research

• Understanding a discipline involves:
  – Building models
    • e.g., application domain, workflows, problem solving processes
  – Checking our understanding is correct,
    • e.g., testing our models, experimenting in the real world
  – Analyzing results to learn, encapsulate knowledge and refine models

• Empirical study of high end computing requires:
  – Scientific use of quantitative and qualitative data to understand and improve the code and code development process
  – Real world laboratories

• In HEC, a development organization might ask:
  – Which development models (MPI, Open MP, Co-Array Fortran) work best in a given environment?
  – Which model will require the least effort for a novice, expert?
  – Which model will provide the best performance?
  – What is the trade-off between development effort and speedup?
Decision-support

• Ultimate goal of the empirical research is to provide guidance to mission partners, vendors, researchers, practitioners about which technologies to choose under what circumstances
• Package knowledge so it can be used to help make decisions
• Integrate models into a framework that provide users with the information they need, e.g., Given a particular context, what approach should a programmer choose?
Stakeholder Needs

• Mission Partners
  – Which parallel programming technologies are most effective for the kinds of problems we work on?
  – What model allows me to increase my effective work staff?
  – How do I shrink the learning curve?

• Vendors
  – Can I confidently demonstrate the elimination or minimization of effort in any workflow steps using my technology?

• Parallel Technology Developers
  – How can I incorporate the study of development time into my work?
  – What technologies help improve development time?
  – What is the tradeoff of development time to speed-up benefits?

• Practitioners
  – What is the best technology available for solving my problem?
Building the knowledge base

• Knowledge is often either:
  – High confidence in a very small region of context space (e.g., classroom experiment)
  – Over a broad context region, but low confidence (e.g., folklore)
• Using meta-analysis, we can combine results from studies
  – What results are consistent across studies, and which ones differ?
  – Can we identify the context variables that are changing across the studies that can account for differences?
  – Can we build useful knowledge across classroom studies, case studies, using hypotheses and validated folklore, providing useful measurements?
• Learning over time
  – Start small – crawl before you walk before you run
  – Evolve studies, models, hypotheses,
Goals: Evolving studies

• Pilot **Classroom Studies** on single programmer assignments
  – Identify variables, data collection problems, single programmer workflows, experimental designs

• Lead to **Observational Studies** on single programmer assignments
  – Develop variables and data we can collect with confidence based upon our understanding of the problems

• Lead to **Controlled** single programmer **experiments**
  – Generate more confidence in the variables, data collection, models, provide hypotheses about novices

• Lead to **team student projects**
  – Study scale-up, multi-developer workflows,

• Lead to **professional developer studies**
  – Study scale-up, multi-developer workflows, porting, reusing, developing from scratch
Goals: Building models

• Identify the **relevant variables** and the context variables, programmer workflows, mechanisms for identifying variables and relationships
  – Developers: Novice, experts
  – Problem spaces: various kernels; computationally-based vs. communication based; …
  – Work-flows: single programmer research model, …
  – Mechanisms: controlled experiments, folklore elicitation, case studies

• Identify what variables can be **collected accurately** or what proxies can be substituted for those variables, understand data collection problems, …

• Identify the **relationship among** those **variables**, and the **contexts** in which those relationships are true

• Build **models** of time to development, relative effectiveness of different programming models, productivity, i.e., Productivity = Value/Cost

• E.g., In the context of a single programmer, productivity might be Relative Speedup/Relative Effort (how do you measure speed-up, how do you measure effort?)
Goals: Evolving Hypotheses

- Identify folklore*: elicit expert opinion to identify the relevant variables and terminology, some simple relationships among variables, looking for consensus or disagreement

- Evolve the folklore: evolve the relationships and identify the context variables that affect their validity, using surveys and other mechanisms

- (Actually we: Identified a set of opinions from a particular set of experts and then tested these opinions against the larger community several times, having modified them after each data collection activity)

- Turn the folklore into hypotheses using variables that can be specified and measured

- Verify hypotheses or generate more confidence in their usefulness in various studies about development, productivity, relative effectiveness of different programming models,
  - E.g., OpenMP offers more speedup for novices in a shorter amount of time when the problem is more computationally-based than communication based.

*Folklore: An often unsupported notion, story, or saying that is widely circulated
Building a body of knowledge

<table>
<thead>
<tr>
<th></th>
<th>Serial</th>
<th>MPI</th>
<th>OpenMP</th>
<th>StarP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest-Neighbor Problems:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game of Life</td>
<td>C3</td>
<td>C0</td>
<td>C1</td>
<td>C3</td>
</tr>
<tr>
<td>Grid of Resist.</td>
<td>C2</td>
<td>C2</td>
<td>C2</td>
<td>C2</td>
</tr>
<tr>
<td>Embarrassingly Parallel Problems:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffon-Laplace Needle</td>
<td>C2</td>
<td>C2</td>
<td>C2</td>
<td>C3</td>
</tr>
</tbody>
</table>

- Validation strategy: If we are building a consistent body of knowledge
  - Similar studies (within a cell) should produce similar results
  - Studies of different phenomena (across rows or columns) should identify measurable influencing factors
Building a body of knowledge: Similar studies, similar results?

<table>
<thead>
<tr>
<th>Serial</th>
<th>MPI</th>
<th>OpenMP</th>
<th>StarP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest-Neighbor Problems:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game of Life</td>
<td>C2</td>
<td>C0 C1</td>
<td>C3</td>
</tr>
<tr>
<td>Grid of Resist</td>
<td>C2</td>
<td>C2</td>
<td>C2</td>
</tr>
<tr>
<td>Embarrassingly Parallel Problems:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffon-Laplace Needle</td>
<td>C2</td>
<td>C2</td>
<td>C3</td>
</tr>
</tbody>
</table>

- On 8 procs, “perfect” speedup = 8
- C0 mean speedup: 4.86 (n=14)
- C1 mean speedup: 4.81 (n=5)
- Means are NOT significantly different (T-test, α=0.05, p=0.96)
- On 8 procs, “perfect” speedup = 8
- C2 mean speedup: 2.01 (n=8)
- C3 mean speedup: 3.73 (n=8)
- Means are NOT significantly different (T-test, α=0.05, p=0.07)
Building a body of knowledge: Differences by problem type?

<table>
<thead>
<tr>
<th></th>
<th>Serial</th>
<th>MPI</th>
<th>OpenMP</th>
<th>StarP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nearest-Neighbor Problems:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game of Life</td>
<td>C3</td>
<td>C0 C1</td>
<td>C3</td>
<td></td>
</tr>
<tr>
<td>Grid of Resist</td>
<td>C2</td>
<td>2</td>
<td>C2</td>
<td>C2</td>
</tr>
<tr>
<td><strong>Embarrassingly Parallel Problems:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffon-Laplace Needle</td>
<td>C2</td>
<td>C3</td>
<td>C2</td>
<td>C3</td>
</tr>
</tbody>
</table>

**MPI, Game of life**
Overall mean speedup: 4.85 (n=19)

Mean speedup IS significantly different for different problems:
(T-test, α=0.05, p=0.006)

**MPI, Buffon-Laplace**
Overall mean speedup: 2.87 (n=16)
Building a body of knowledge: Differences by HPC model?

Game of life: **MPI more costly**

Effort (hours)
- MPI: 9.1 (n=15)
- OpenMP: 4.3 (n=15)
- Effort(MPI) IS sig. greater than effort(OMP) (Paired t-test, α=0.05, p=0.01)

<table>
<thead>
<tr>
<th></th>
<th>Serial</th>
<th>MPI</th>
<th>OpenMP</th>
<th>StarP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nearest-Neighbor Problems:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Game of Life</td>
<td>C0</td>
<td>C1 C0</td>
<td>C3 C3</td>
<td></td>
</tr>
<tr>
<td>Grid of Resist</td>
<td>C2</td>
<td>C2</td>
<td>C2</td>
<td>C2</td>
</tr>
<tr>
<td><strong>Embarrassingly Parallel Problems:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Buffon-Laplace Needle</td>
<td>C2</td>
<td>C2</td>
<td>C2</td>
<td>C3</td>
</tr>
</tbody>
</table>

**Buffon-Laplace: OpenMP more costly**

Effort (hours)
- MPI: 1.4 (n=20)
- OpenMP: 2.5 (n=20)
- Effort(OMP) IS sig. greater than effort(MPI) (Paired t-test, α=0.05, p=0.02)
Building a body of knowledge: Looking across studies

Effect of model & problem on % Effort saved using OpenMP instead of MPI

Performance by problem/HPC model

Effort

Performance
Summary

• Start small
  – Run small studies
  – Use kernels: small problems - the building blocks of larger parallel programs
  – Use small data sets so that programs don’t take long to run
• Collect data on programmers already working on parallel programs
  – Students in parallel programming classes
  – Case studies of real programming projects
• Leverage expertise of HEC community
  – Gather folklore through focus groups, surveys
  – Review of existing literature on parallel programming issues
• Build knowledge to support decision-making in HEC
  – Build individual pieces of knowledge through empirical studies
  – Assemble pieces of knowledge through meta-analysis
  – Package results in decision-support system
Threats to Validity

• Internal
  – Learning
  – Effort data instrumentation
  – Performance data
  – Incomplete data

• External
  – Size of problem assignments
  – Experience
Conclusions

- Effect of HPC programming model is large enough to measure in classroom studies despite small sample sizes
- OpenMP saves development effort vs. MPI on many problems
  - Although not under all conditions
- Difficult to analyze effect on performance; so far, too many sources of variation
  - Implies “lab packages” are needed to
    - specify the *essential invariabilities* across multiple classes
    - while still allowing instructors freedom in their local environment
Phase I: 2004-2006 activities

Variables and Technologies

Classroom Studies

Results

Phase II: 2005-2007 activities

ACSI studies

Results

Variables and Technologies

Phase III: 2006-2008 activities

Use of classroom study results

- Classroom empirical evidence about HPC development time issues
- Focus academics on development time issues
- Develop effective measures
- Evolve techniques for teaching parallel programming

Compare

- Industrial empirical evidence about HPC development time issues
  WHERE classroom and ACSI studies the same
  - Use classroom as proxy for industrial environment
  WHERE not same
  - Evolve classroom studies
  - Do more ACSI and MP studies
Thanks to…

- This research would not be possible without the HPC professors:
  - Alan Edelman
  - John Gilbert
  - Mary Hall & Jacqueline Chame
  - Ed Luke
  - Aiichirio Nakano
  - Allan Snavely & Henri Casanova
  - Alan Sussman
  - Uzi Vishkin
  - And collaborators: Marv Zelkowitz, Jeff Carver, Lorin Hochstein, Forest Shull, Sima Asgari, Jeff Hollingsworth, …
For More Info

- HPCS program homepage
  - http://www.highproductivity.org/

- Lab package for experimenters
Clearinghouse Project

Problem: How do I pick the right set of processes for my environment.

Project Goal: Populate an experience base for acquisition best practices, defining context and impact attributes allowing users to understand the effects of applying the processes based upon the best empirical evidence available.

Research Goal: Define a repeatable model-based empirical evidence vetting process enabling different people to create profiles consistently and the integration of new evidence.

Partners: OSD, UMD, FC-MD, DAU, CSC, …
Operational Concept

Best Practice Handling

Identification | Quantification & Qualification | Analysis & Synthesis | Validation | Packaging & Dissemination

Information Handlers

Best Practice Contributions

Support Team

Information Providers

BPCh IT Components

Best Practice Handling

System Administration

Repository

Intelligent Front-ends

BPCh Operations

Initial Development | System Upgrades & Maintenance | Backups & User Management

BPCh Usage

Information request | Project characterization | Access data | Select Appropriated Practice | Interface with other resources

Information Seekers

BPCh Process Components | BPCh Roles | BPCh IT Components | BPCh Role Specific Interfaces
BPCh recommendations based on evidence from real programs.

Evidence

• **Source**: How trustable?

• **Context**: Used by a safety critical program? In a DoD environment? On a warfighter?

• **Results**: Did it *increase* or *reduce* cost, quality, and schedule?
Summary

The summary says where the practice was successful what it helped and cost how to get started

Practices are vetted for accuracy and usefulness

Evidence 1
Source
Context
Results

Evidence 2
Source
Context
Results

Evidence 3
Source
Context
Results

Evidence 4
Source
Context
Results
Help me find a practice to reduce schedule.

Who's used it for safety critical programs?

Summary

Evidence 1
Source
Context
Results

Evidence 2
Source
Context
Results

Evidence 3
Source
Context
Results

Evidence 4
Source
Context
Results

Acquisition manager, safety critical program
Summarizing

• Measurement is fundamental to any engineering science

• User needs must be made explicit (measurable models)

• Organizations have different characteristics, goals, cultures; stakeholders have different needs

• Process is a variable and needs to be selected and tailored to solve the problem at hand

• We need to learn from our experiences, build software core competencies

• Interaction with various industrial, government and academic organizations is important to understand the problems

• To expand the potential competencies, we must partner
Where do we need to go?
Propagating the empirical discipline

Build an empirical research engine for software engineering

• Build testbeds for experimentation and evolution of processes

• Build product models that allow us to make trade-off decisions

• Build decision support systems offering the best empirical advice for selecting and tailoring the right processes for the problem

• Use empirical study to test and evolve technologies for their appropriateness in context