

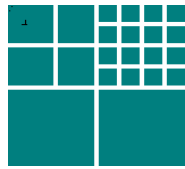
Empirical Software Engineering

Where are we headed?

Victor R. Basili

**University of Maryland
and**

Fraunhofer Center - Maryland



Why Empirical Software Engineering?

Understanding a discipline involves **building models**,
e.g., application domain, problem solving processes

And checking our understanding is correct,
e.g., testing our models, **experimenting** in the real world

Analyzing the results involves **learning**, the **encapsulation of knowledge** and the ability to change or refine our models over time

The understanding of a discipline evolves over time

This is the **empirical paradigm** that has been used in many fields, e.g., physics, medicine, manufacturing

Like other disciplines, **software engineering** requires an empirical paradigm



Why Empirical Software Engineering?



Empirical software engineering requires the scientific use of quantitative and qualitative data to understand and improve the software product, software development process and software management

It requires real world laboratories

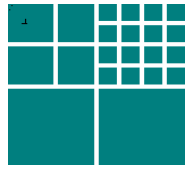
Research needs laboratories to observe & manipulate the variables

- they only exist where developers build software systems

Development needs to understand how to build systems better

- research can provide models to help

Research and Development have a **symbiotic relationship** requires a working relationship between industry and academe



Where are we today?

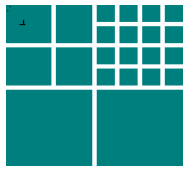
“Under specified conditions, ...”

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)

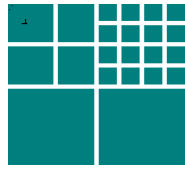
Technique Definition Guidance

- 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)



What do I believe?

- 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
- Believing this **changes how we think**, what we do, what is important

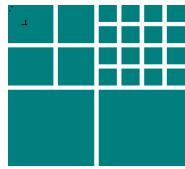


What do I believe?

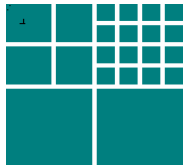
- Measurement is fundamental to any engineering science
- User needs must be made explicit (measurable models)
- Organizations have different characteristics, goals, cultures; stakeholders have different needs
- Process is a variable and needs to be selected and tailored to solve the problem at hand
- 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?

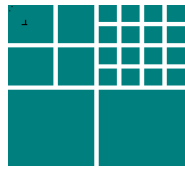


- Propagate the empirical discipline
- Build an empirical research engine for software engineering
- Build testbeds for experimentation and evolution of processes
- Build product models that allow us to make trade-off decisions
- Build decision support systems offering the best empirical advice for selecting and tailoring the right processes for the problem



What am I doing now?

- Consider the following projects:
 - CeBASE – Center for Empirically-Based Software Engineering
 - HDCP – High Dependability Computing Project
 - HPCS – High Productivity Computing Study
 - Best Practices Clearinghouse



CeBASE

Center for Empirically Based Software Engineering

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

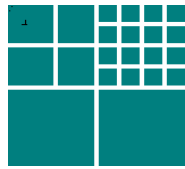
CeBASE Research Goal: Create and evolve an **empirical research engine** for building the research methods that can provide the empirical evidence of what works and when

Partners: UMD, FC-MD, USC, UNC, ...





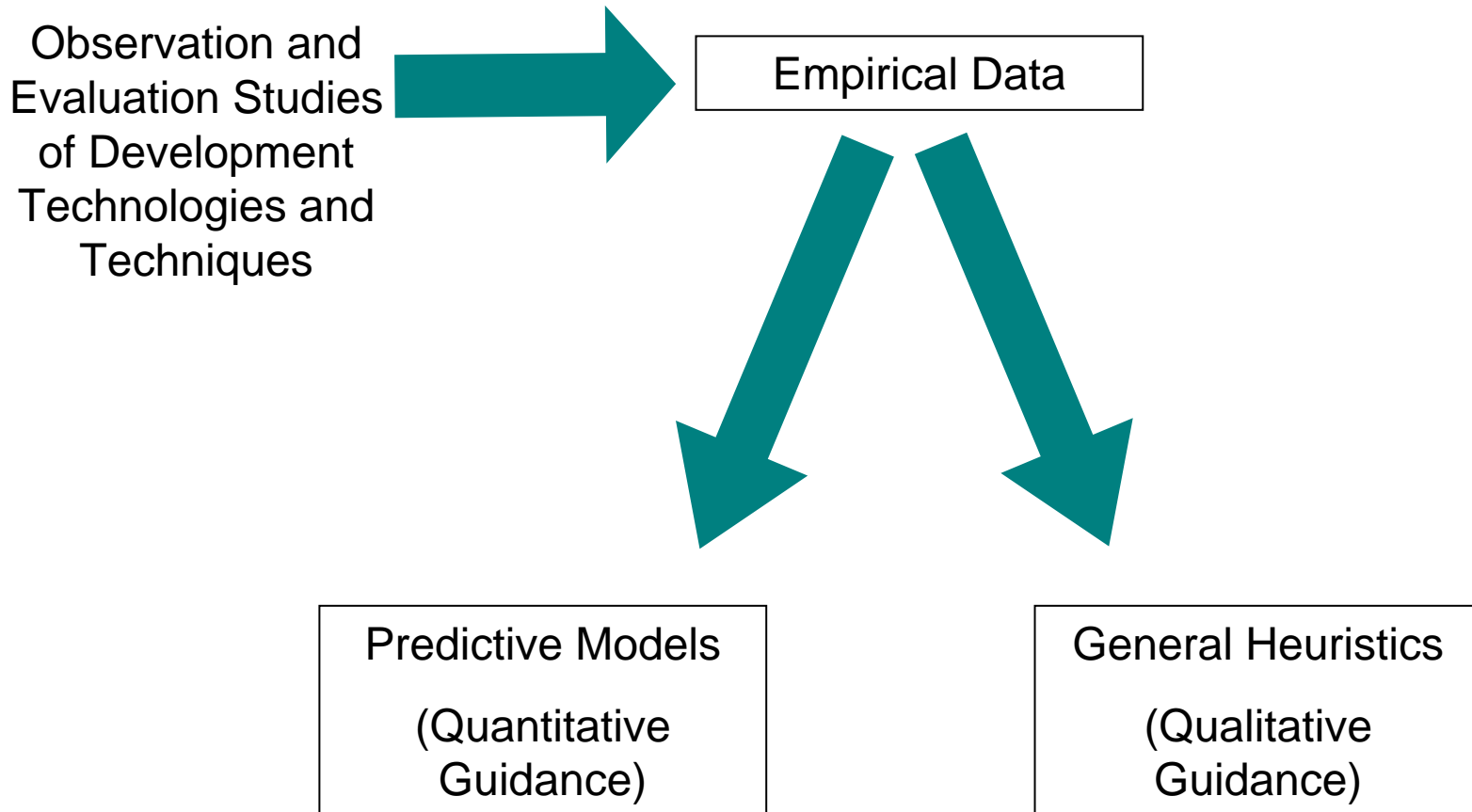
CeBASE Empirical Research Engine



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

CeBASE Approach

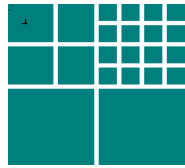


E.g. COCOTS excerpt:

Cost of COTS tailoring = $f(\# \text{ parameters initialized, complexity of script writing, security/access requirements, ...})$

E.g. Defect Reduction Heuristic:

For faults of **omission** and **incorrect specification**, **peer reviews** are more effective than functional testing.



NASA High Dependability Computing Program

Project Goal: Increase the ability of NASA to engineer highly dependable software systems via the development of new techniques and technologies

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

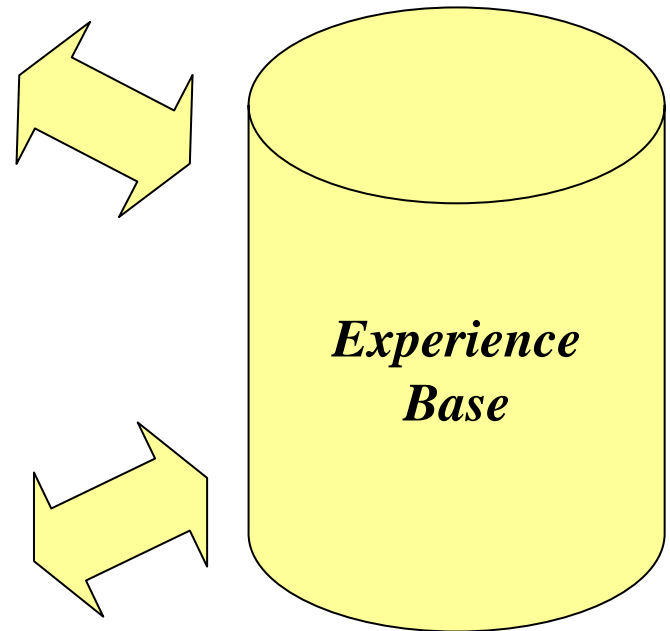
Main Questions



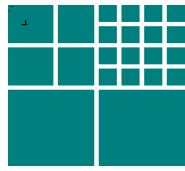
System Developers

How can I understand the stakeholders quality needs?

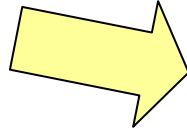
How can I apply the available processes to deliver the required quality?



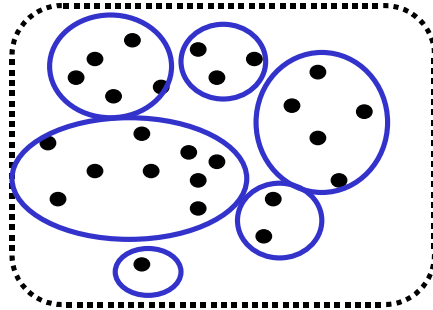
What are the top level research problems?



System Users



Failures Space

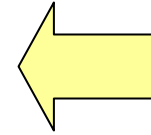
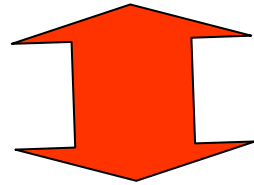


Research Problem 3

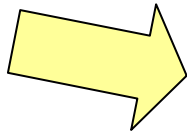
What set of processes should be applied to achieve the desired quality? (Decision Support)

Research Problem 1

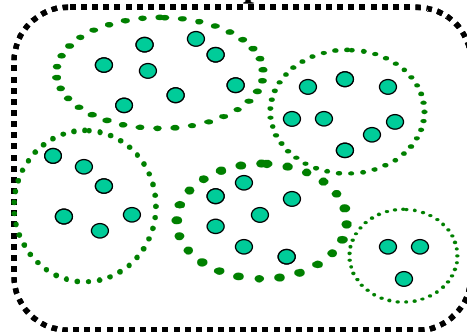
Can the quality needs be understood and modeled? (Product Models)



Process Developers



Fault Space

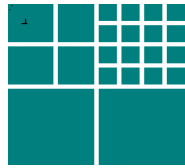


System Developers

Research Problem 2

What does a process do?

Can it be empirically demonstrated?



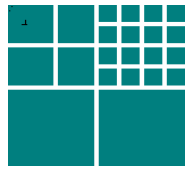
System User Issues

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

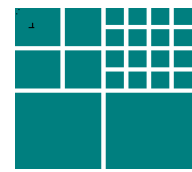
- How do I identify the non-functional requirements in a consistent way?
 - Across multiple stakeholders
 - In a common terminology (Failure focused)
 - Able to be integrated
- How can I take advantage of previous knowledge about failures relative to system functions, models and measures, reactions to failures?
 - Build an experience base
- How do I identify incompatibilities in my non-functional requirements for this particular project?



UMD - Unified Model of Dependability



- 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/....)

concern

cause

issue

FAILURE	HAZARD
<ul style="list-style-type: none"> - Type <ul style="list-style-type: none"> - Accuracy - Response Time - etc. - Availability impact <ul style="list-style-type: none"> - Stopping - Non-Stopping - Severity <ul style="list-style-type: none"> - High - Low 	<ul style="list-style-type: none"> - Severity <ul style="list-style-type: none"> - People affected - Property only - etc.

manifest

trigger

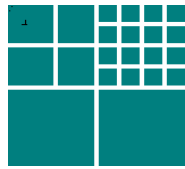
event

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

reaction

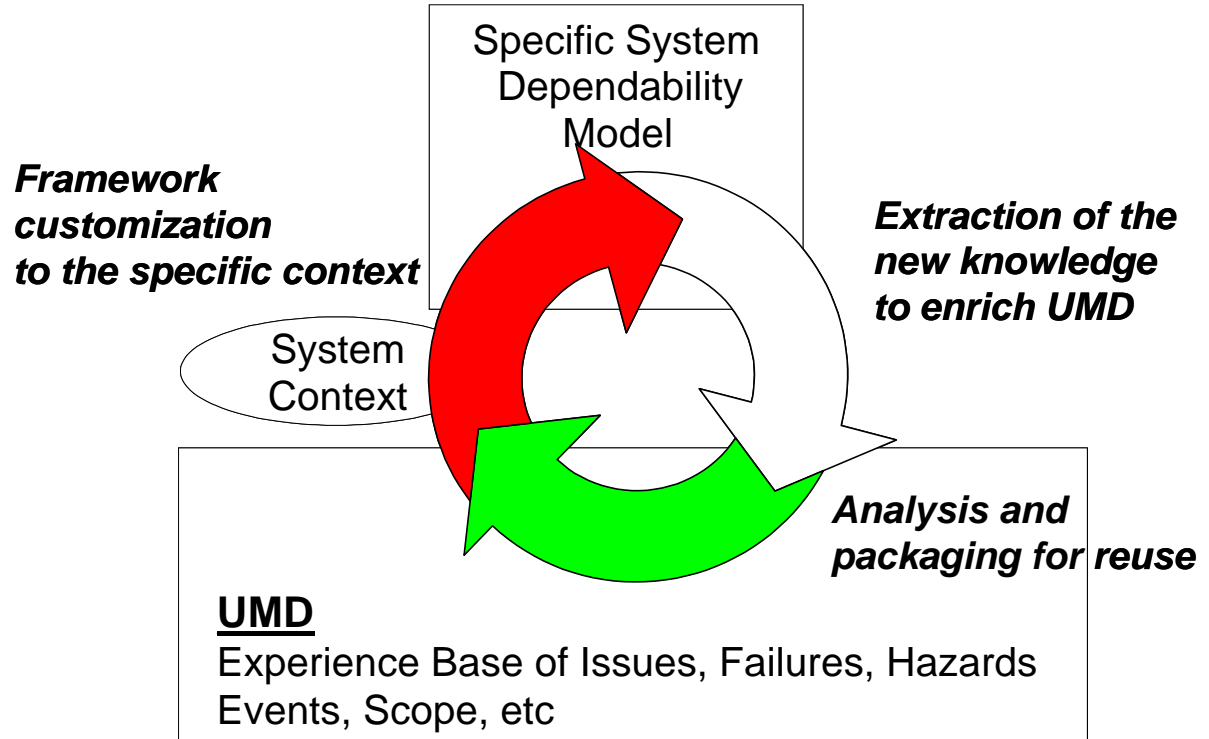
- **Impact mitigation**
 - warnings
 - alternative services
 - mitigation services
- **Recovery**
 - recovery time / actions
- **Occurrence reduction**
 - guard services

UMD assimilates new experience



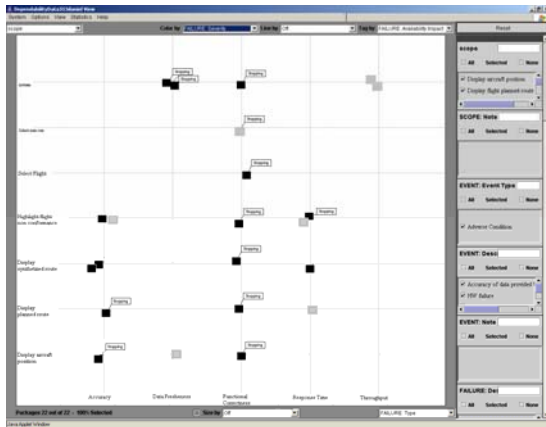
Characterizations (e.g., types, severity, etc.) of the basic UMD modeling concepts of issue, scope, measure, and event depend on the specific context (project and stakeholders).

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

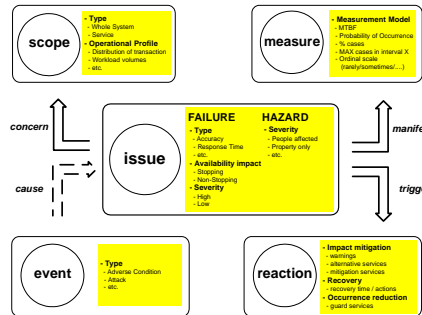
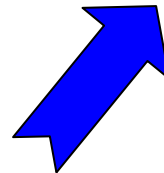
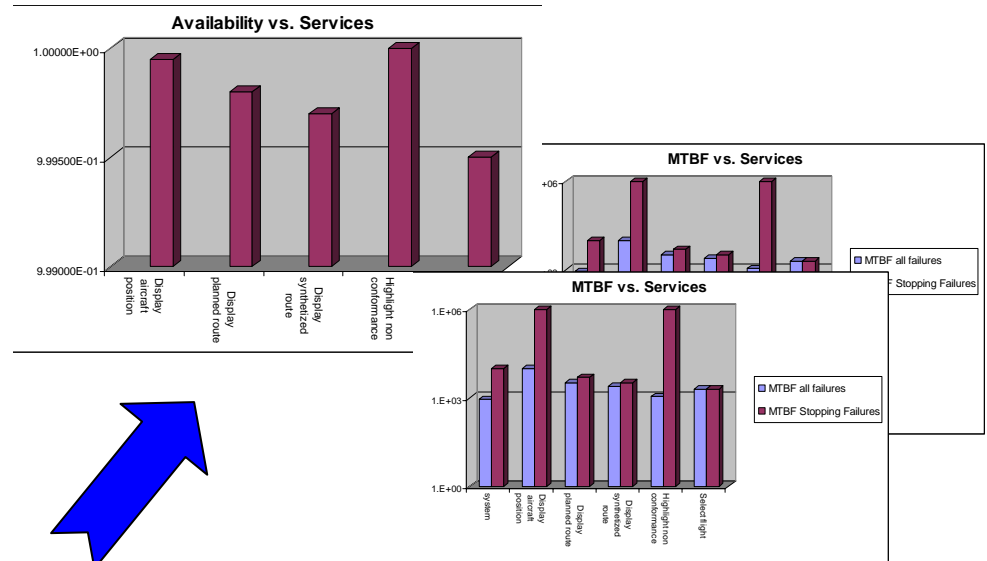
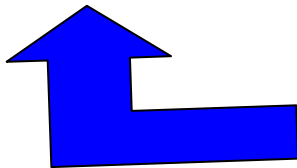




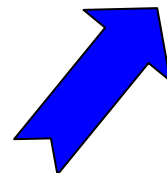
UMD support engineering decisions at requirements phase for quality validation, negotiation, trade-offs analysis

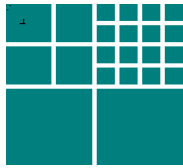


Requirements Visualization



Computation of aggregate values of dependability (availability, MTBF per service, etc)





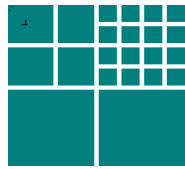
Technology Researcher Issues

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

- How does one articulate the goals of a technology?
 - Formulating measurable hypotheses
- How does one empirically demonstrate its goals?
 - Performing empirical studies
 - Validate expectations/hypotheses
- What are the requirements for a testbed?
 - Fault seeding
- How do you provide feedback for improving the technology?



Using testbeds to transfer technology



- **Define Testbeds**

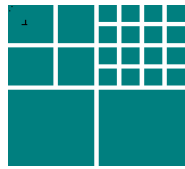
- Projects, operational scenarios, detailed evaluation criteria representative of product needs
- Stress the technology and demonstrate its context of effectiveness
- Help the researcher identify the strengths, bounds, and limits of the particular technology at different levels
- Provides insight into the integration of technologies
- Reduce costs by reusing software artifacts
- Reduce risks by enabling technologies to mature before taking them to live project environments
- Assist technology transfer of mature results

- **Conduct empirical evaluations** of emerging processes

- Establish evaluation support capabilities: instrumentation, seeded defect base; experimentation guidelines



TSAFE testbed:



Tactical Separation Assisted Flight Environment

- Aids air-traffic controllers in detecting short-term aircraft conflicts
- Proposed as principle component of larger Automated Airspace Computing System by Heinz Erzberger at NASA Ames
- MIT TSAFE: a partial implementation by Gregory Dennis
 - Trajectory synthesis and Conformance Monitoring
- TSAFE Testbed: developed at FC-MD, based on MIT TSAFE
 - Added testbed specific features, e.g. to monitor faults, output
 - Added features to make it easier to run and experiment with Testbed
 - Synthesized faults that were seeded
 - Added documentation
 - Added functionality, algorithms
- Used to evaluate a family of software architecture techniques



Testbed Development: Fault Seeding



Fault Seeding Drivers

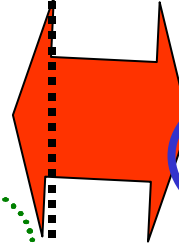
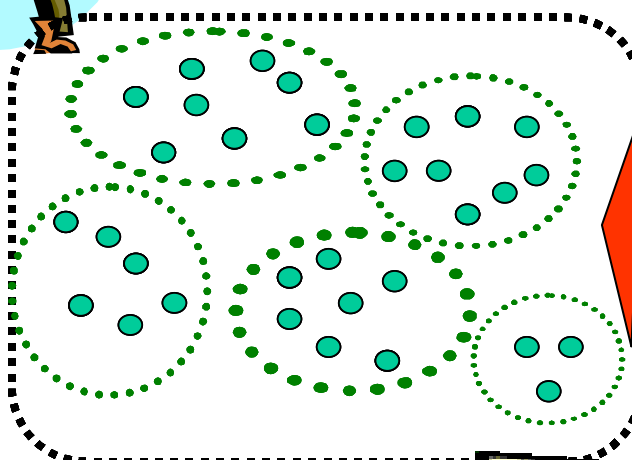
- **Intervention-driven:** seed faults based upon what the intervention is expected to find
 - Assumes interventionist can express goals in terms of fault classes
 - Focus on intervention (no direct dependability link)
 - Each intervention can generate a different set of faults
- **History-driven:** seed faults based upon the fault history from similar NASA projects
 - Assumes we have historical data from similar systems
 - Maybe we have the relationship of failure to fault
- **Dependability-driven:** seed faults that create issues (failures) that the user sees as hindering dependability
 - Assumes we can identify issues that would impact stakeholders needs
 - Fault may not be representative of what caused the failure

Approaches can be combined

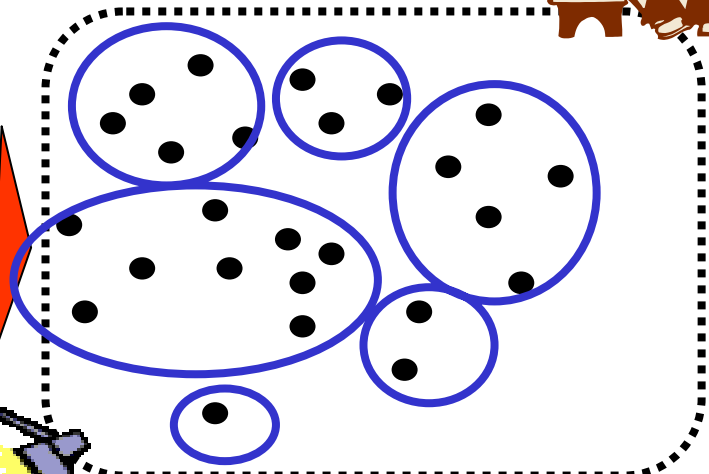
Problem: Satisfying dependability needs by applying the right interventions: Matching Failures Classes & Faults Classes

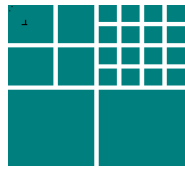


Faults Space



Failures Space





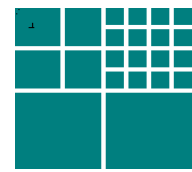
High Productivity Computing Systems

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



HPCS Phase II Teams

➤ Create a new generation of **economically viable computing systems** and a **procurement methodology** for the security/industrial community (2010)

Industry

PI: Elnozahy

PI: Mitchell

PI: Smith

Mission Partners



Office of Science
U.S. Department of Energy



HPC Modernization Program
National Nuclear Security Administration

Productivity Team (Lincoln Lead)

PI: Kepner

PI: Lucas

PI: Basili

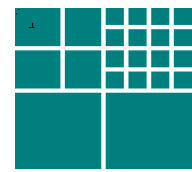
PI: Benson & Snively

PI: Dongarra

PI: Koester

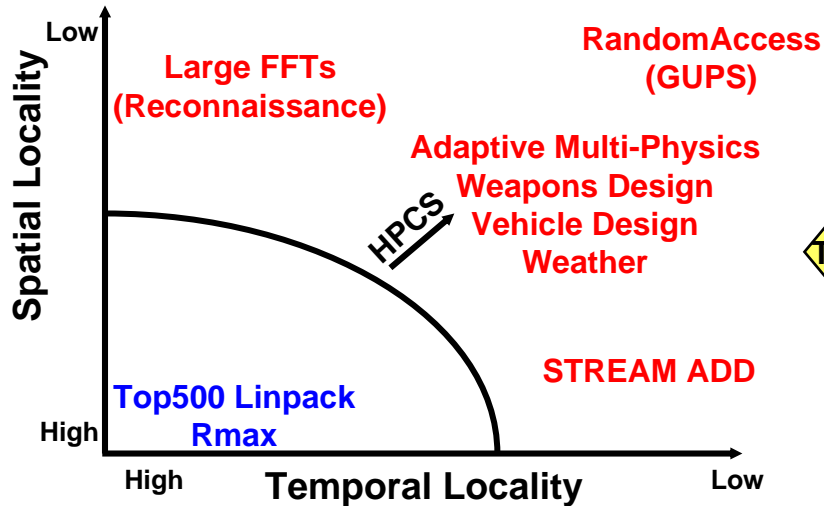
Los Alamos National Laboratory
ARGONNE
BERKELEY LAB
PIs: Vetter, Lusk, Post, Bailey

UCSB
CSAIL
Ohio State
CODESOURCERY
PIs: Gilbert, Edelman, Ahalt, Mitchell

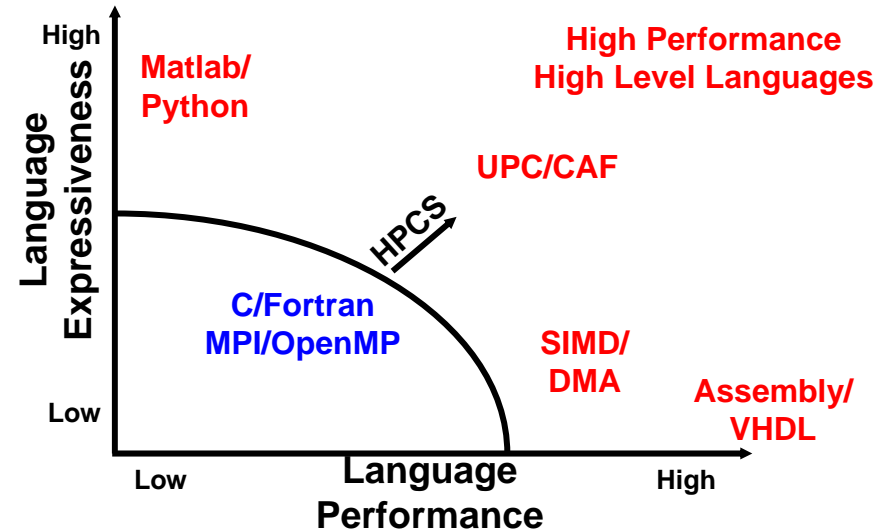


Trade-offs Drive Designs

Execution Time (Example)



Development Time (Example)



Current metrics favor caches and pipelines

- Systems ill-suited to applications with
- Low spatial locality
- Low temporal locality

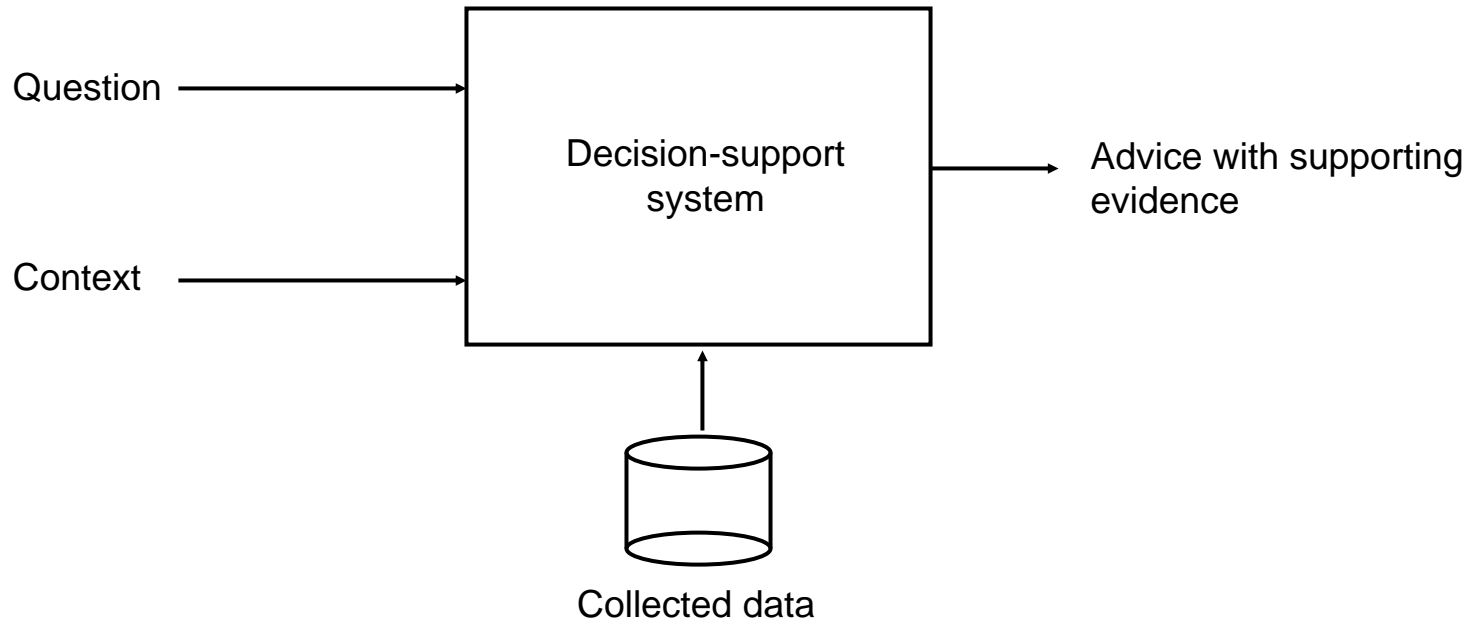
No metrics widely used

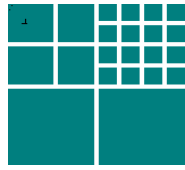
- Least common denominator standards
- Difficult to use
- Difficult to optimize

- HPCS needs a validated assessment methodology that values the “right” vendor innovations (see HPCchallenge)
- Allow **tradeoffs** between Execution and Development Time

Decision-support

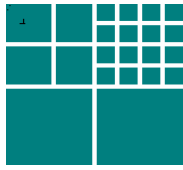
- Ultimate goal of the empirical research is to provide guidance to mission partners, vendors, researchers, practitioners about which technologies to choose under what circumstances
- Package knowledge so it can be used to help make decisions
- Integrate models into a framework that provide users with 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?



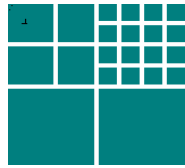
Building the knowledge base

- 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,



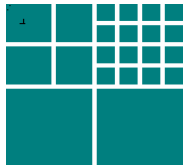
Goals: Evolving studies

- 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., $\text{Productivity} = \text{Value}/\text{Cost}$
- 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?)**

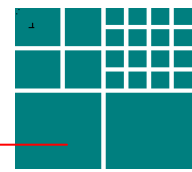


Goals: Evolving Hypotheses

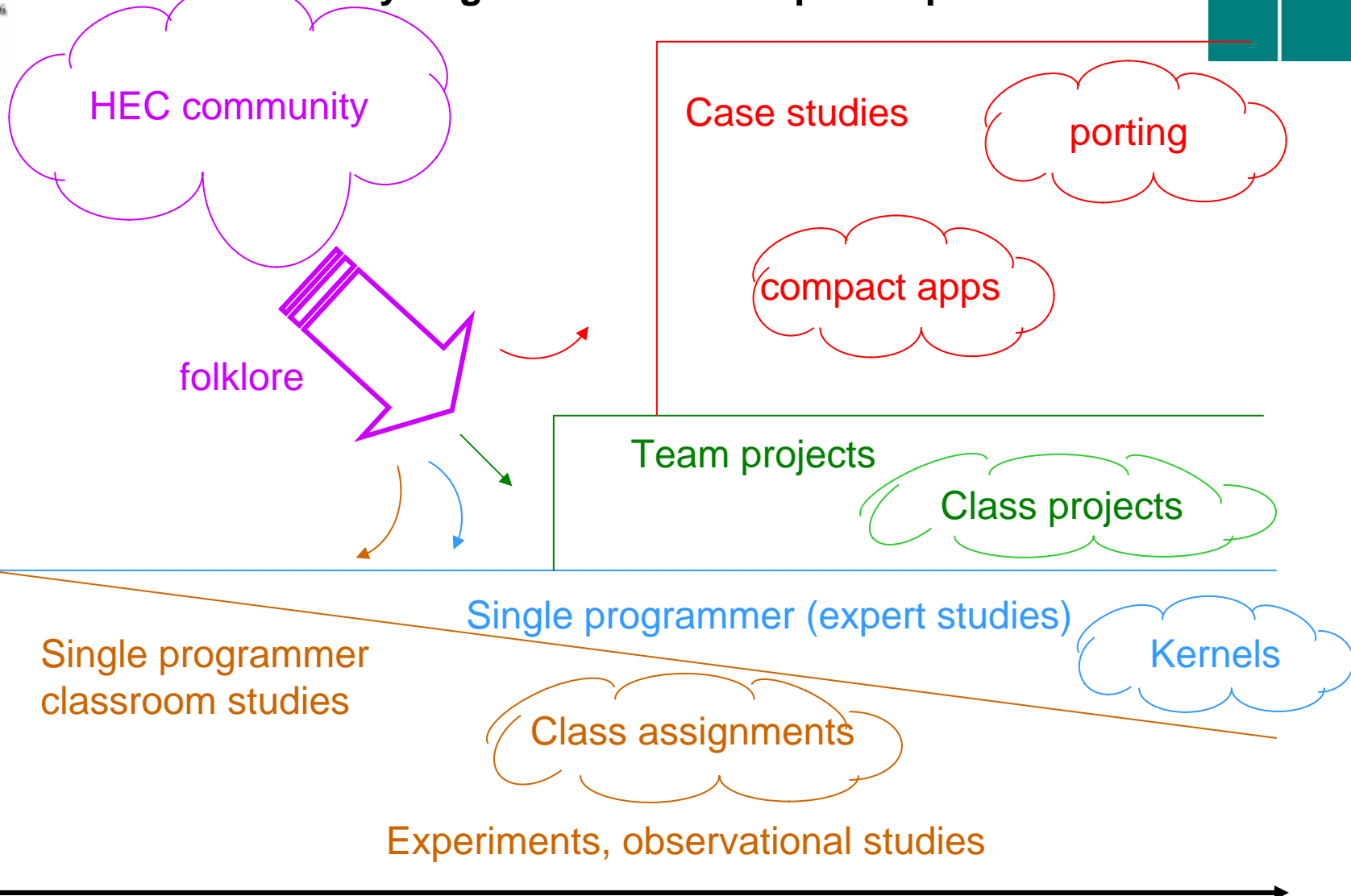
- Identify folklore*: elicit expert opinion to identify the relevant variables and terminology, some simple relationships among variables, looking for consensus or disagreement
- Evolve the folklore: evolve the relationships and identify the context variables that affect their validity, using surveys and other mechanisms
- (Actually we: Identified a set of opinions from an 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

HEC community provides questions to study that lead to successively larger and more complex experiments



Problem Scale

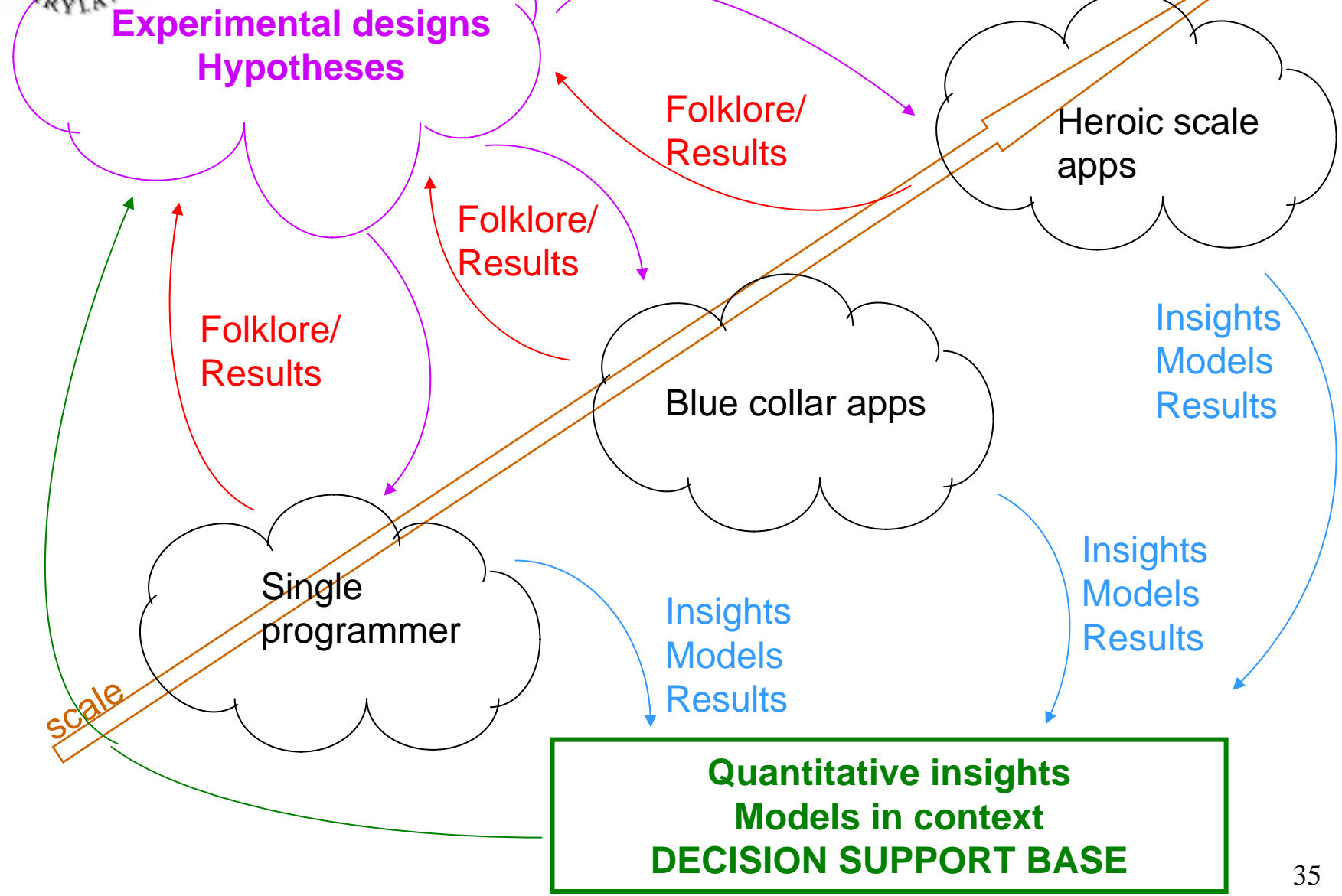
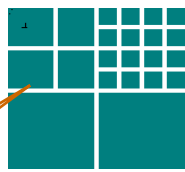


Time/Duration

Evolving measurement, models, hypotheses



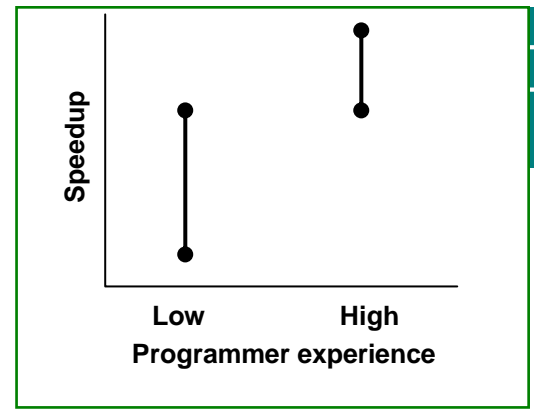
Experiments are used to build a decision support base of results



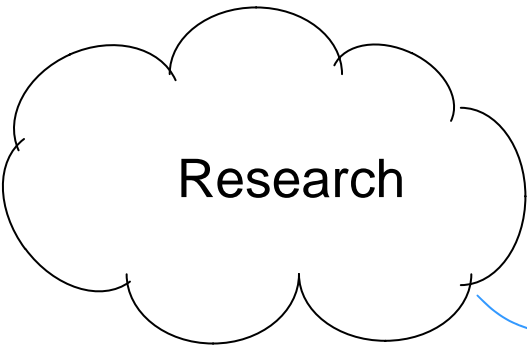


The results of the experiments provide a variety of models and relationships useful for understanding HEC workflows

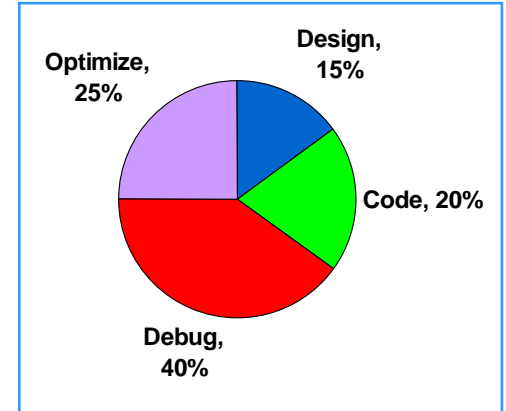
Relationships among variables



Program performance

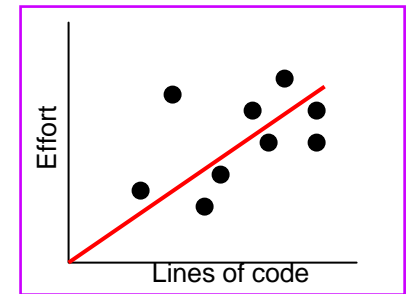


Baseline data

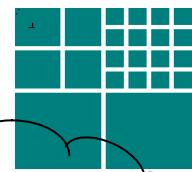
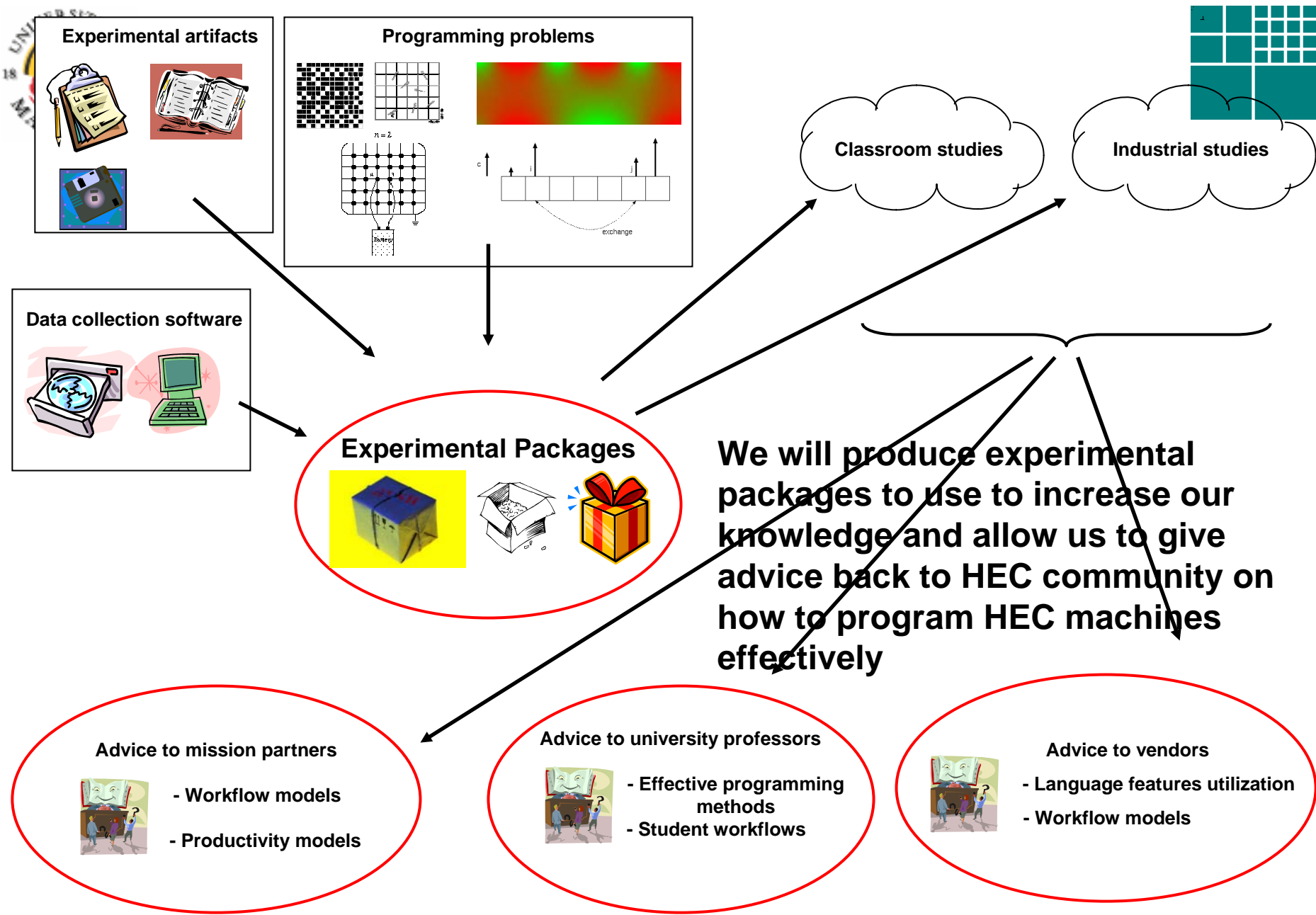


Workflows

Evaluation of measures




Productivity 36




We will produce experimental packages to use to increase our knowledge and allow us to give advice back to HEC community on how to program HEC machines effectively

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

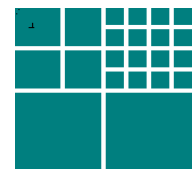


OSD Clearinghouse Project

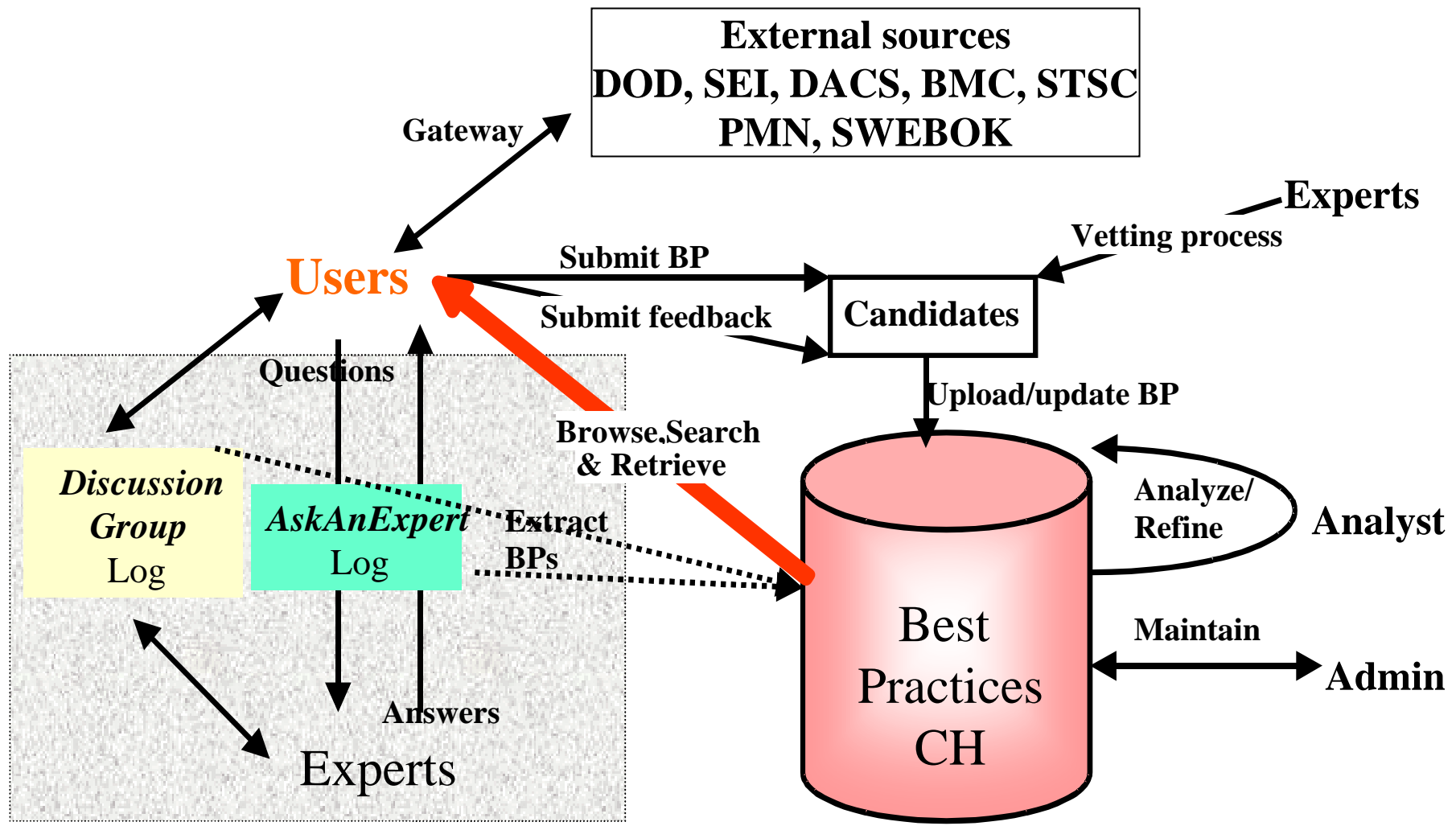
Project Goal: Populate an experience base for acquisition best practices, defining **context and impact** attributes allowing users to understand the effects of applying the processes based upon the best empirical evidence available

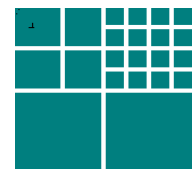
Research Goal: Define a repeatable model-based empirical evidence vetting process enabling different people to create profiles consistently and the integration of new evidence

Partners: OSD, UMD, FC-MD, DAU, Northup-Grumman, ...



Clearinghouse Key Concepts





Best Practices Vetting Process

Each cycle allows more experience to be gathered and processed, leading to better characterization of the practice, improved recommendations, and more dependable implementation guidance.



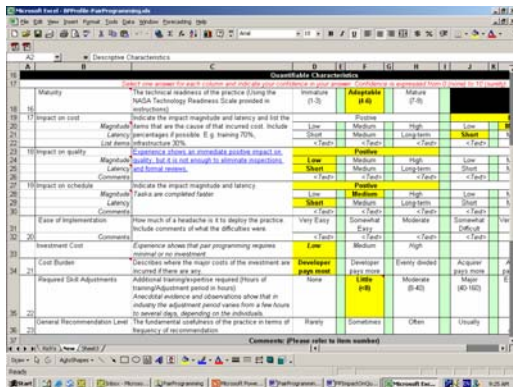
Identification	Characterization	Analysis & Synthesis	Validation	Packaging & Dissemination
<p>Inputs: Leads to practices</p> <p>Activities:</p> <ul style="list-style-type: none"> •Collect •Categorize •Filter •Synthesize •Prioritize <p>Outputs: Candidate set of practices</p>	<p>Inputs: Set of candidate practices and rationale for consideration</p> <p>Activities:</p> <ul style="list-style-type: none"> •Gather/research characteristics about the practice including context (project, etc.), evidence of use, lessons learned •Complete “story” profile <p>Outputs: More detailed set of candidate practices with “stories”</p>	<p>Inputs: Detailed set of candidate practices</p> <p>Activities:</p> <ul style="list-style-type: none"> •Aggregate stories, create profile of practice •Populate the repository •Identify/define Interrelationships <p>Outputs: Single profile for each best practice, associated artifacts, and confidence levels</p>	<p>Inputs: Sets of practice data; validation criteria</p> <p>Activities:</p> <ul style="list-style-type: none"> •Check outputs from previous phases •Color Code practices •Approve practices via panel of experts <p>Outputs: Validated practices</p>	<p>Inputs: Sets of practice data; validation criteria</p> <p>Activities:</p> <ul style="list-style-type: none"> •Packaging •Publishing •Promoting •Providing user help •Discussions <p>Outputs:</p> <ul style="list-style-type: none"> •Repository update •Papers & conference presentations •Course materials/updates

- Proven
- Consistent results
- Initial validation
- Nominated

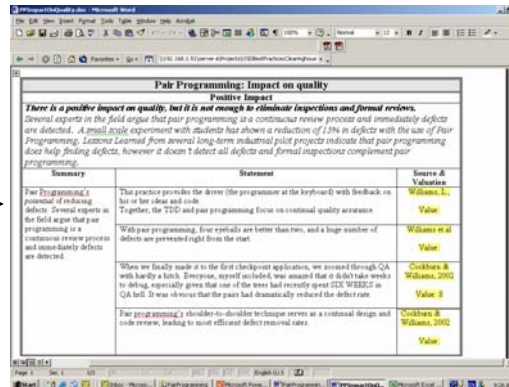
Possible practice validation coding

Empirically-Based Practices

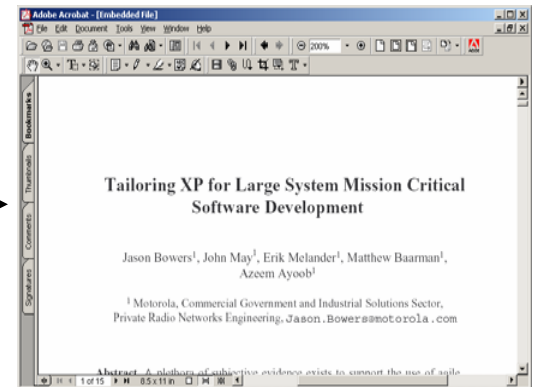
- **Profile**
 - Attributes, Values, Brief justification, links to
 - **Empirical evidence**
 - Justification, Summary, Statement, Source, Valuation, links to
 - **Sources**
 - (Full report/paper, Summary/Story)
- Need models for evaluating the maturity of best practices, weighting empirical evidence



Attribute	Low	Medium	High	Very High
Maturity	Low	Medium	High	Very High
Impact on cost	Low	Medium	High	Very High
Impact on quality	Low	Medium	High	Very High
Impact on schedule	Low	Medium	High	Very High
Ease of implementation	Very Easy	Easy	Medium	Somewhat Difficult
Investment Cost	Low	Medium	High	Very High
Cost Benefit	Low	Medium	High	Very High
Required QA Adjustments	None	Minor	Moderate	Major
General Recommendation Level	Rarely	Sometimes	Often	Usually



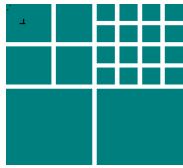
Summary	Statement	Source & Valuation
<p>The programing is potential of reducing defects. Several experts in the field argue that pair programming or a continuous review process and immediately defects are detected.</p>	<p>The practice provides the driver (the programmer at the keyboard) with feedback on his or her ideas and code. Together, the TDD and pair programming focus on continual quality assurance.</p> <p>With pair programming, four eyeballs are better than two, and a larger number of defects are prevented right from the start.</p>	<p>Wilsons, 2002</p> <p>Value: I</p>
<p>When we finally made it to the first checkpoint application, we learned through QA with help by a third developer, myself included, we learned that it didn't take weeks to debug, especially given that one of the team had recently spent SIX WEEKS in QA hell. It was obvious that the pairs had dramatically reduced the defect rate.</p>		<p>Cookles & Wilsons, 2002</p> <p>Value: II</p>
<p>Pair programming is a shoulder-to-shoulder to change serves as a continual design and code review, leading to most efficient defect removal rates.</p>		<p>Cookles & Wilsons, 2002</p> <p>Value: III</p>



Tailoring XP for Large System Mission Critical Software Development

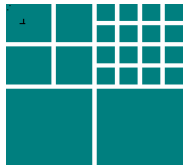
Jason Bowers¹, John May¹, Erik Melander¹, Matthew Baarman¹, Azem Ayoub¹

¹ Motorola, Commercial Government and Industrial Solutions Sector, Private Radio Networks Engineering, Jason.Bowers@motorola.com



Areas of Research Needed

- Eliciting quality requirements in measurable terms
- Evaluating technology in context
- Building testbeds to support experimentation
- Building heuristics to deal with the fault/failure relationship
- Building decision support systems
- Finding better ways to experiment and integrate the results of the studies



Conclusion

- The **synergistic relationship between research, applied research, and practice**
→ **collaborations among multiple groups**
- Software developers need to know what works and under what circumstances
- Technology developers need feedback on how well their technology works and under what conditions
- We need
 - to continue to collect empirical evidence
 - analyze and synthesize the data into models and theories
 - Collaborate to evolve software engineering into an engineering discipline