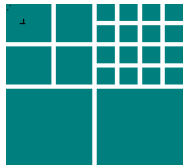


The Role of Empirical Study in Software Engineering

Victor R. Basili

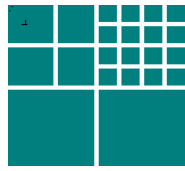
**University of Maryland
and**

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



Motivation for Empirical Software Engineering

Understanding a discipline involves **observation, model building,** and **experimentation**

Learning involves the encapsulation of “knowledge”, checking our “knowledge” is correct, and evolving it over time

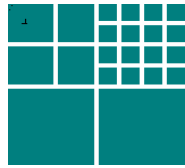
This is the **empirical paradigm** that has been used in many fields, e.g., physics, medicine, manufacturing

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

The nature of the field influences the approach to empiricism.



Motivation for Empirical Software Engineering



Empirical software engineering involves 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

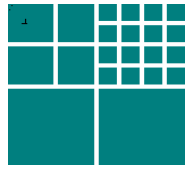
Research needs laboratories to observe & manipulate the variables

- 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 **synergistic relationship** that requires a working relationship between industry and academe



Motivation for Empirical Software Engineering

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)

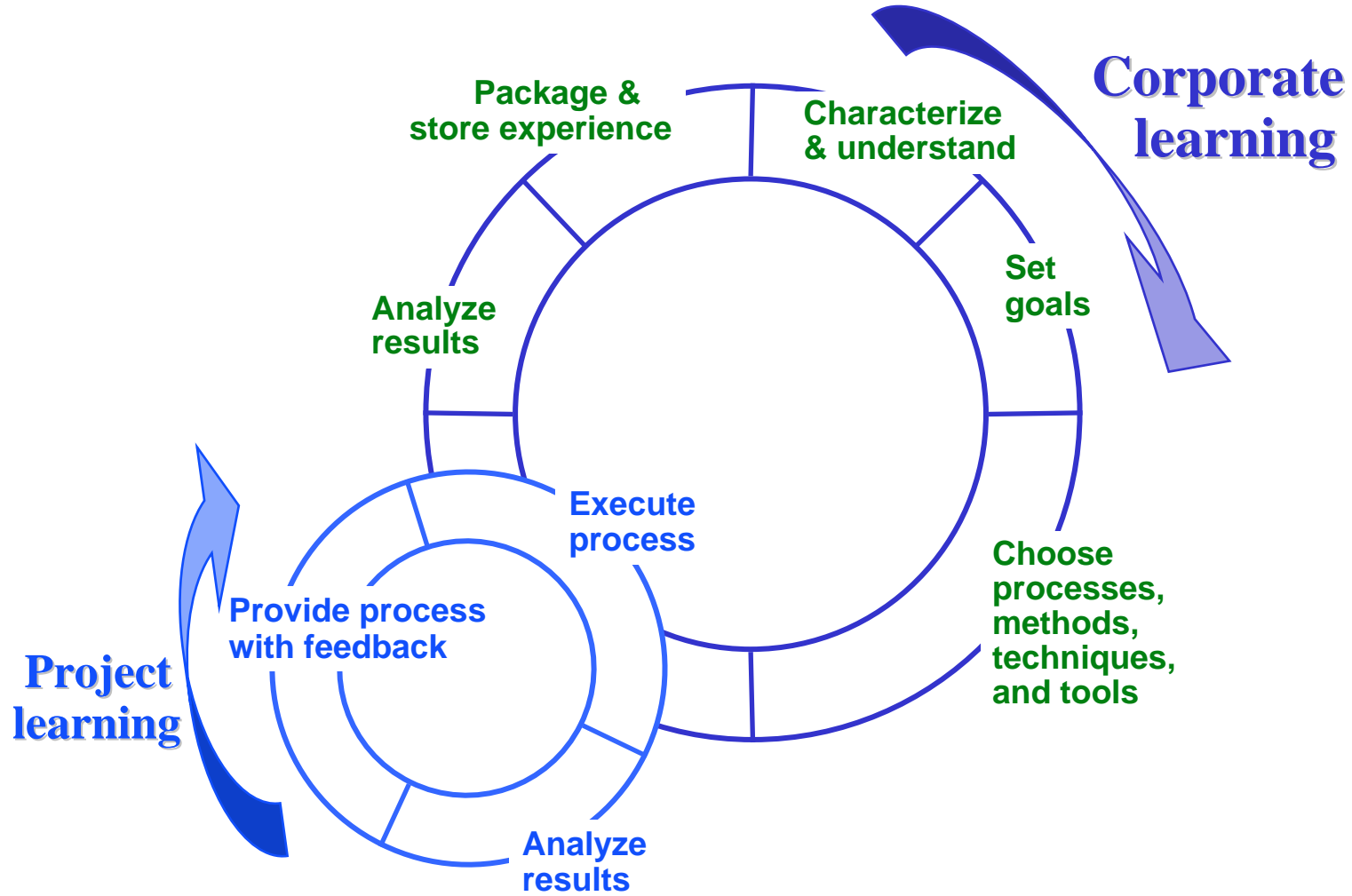
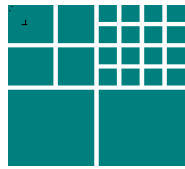
An approach for establishing project and corporate goals and a mechanism for measuring against those goals

Experience Factory (EF)

An organizational approach for building software competencies and supplying them to projects

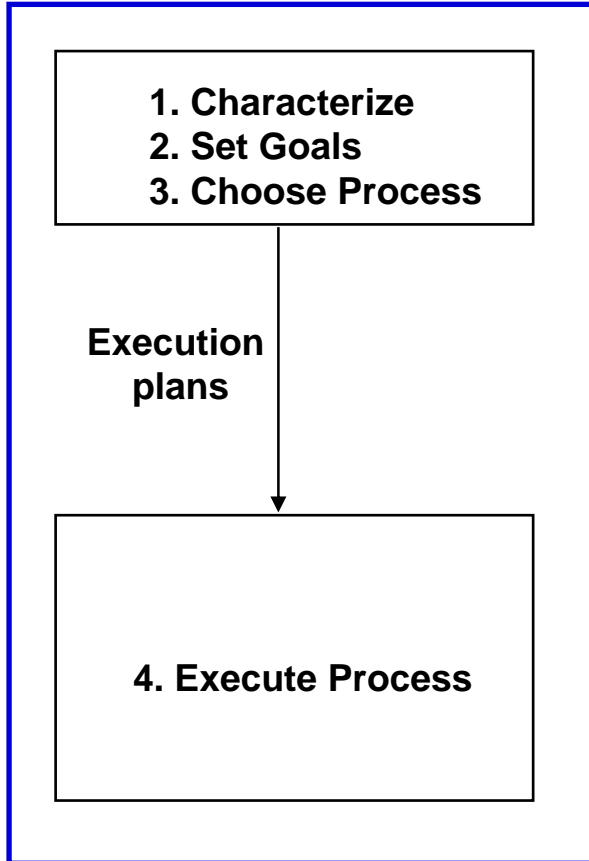


Quality Improvement Paradigm

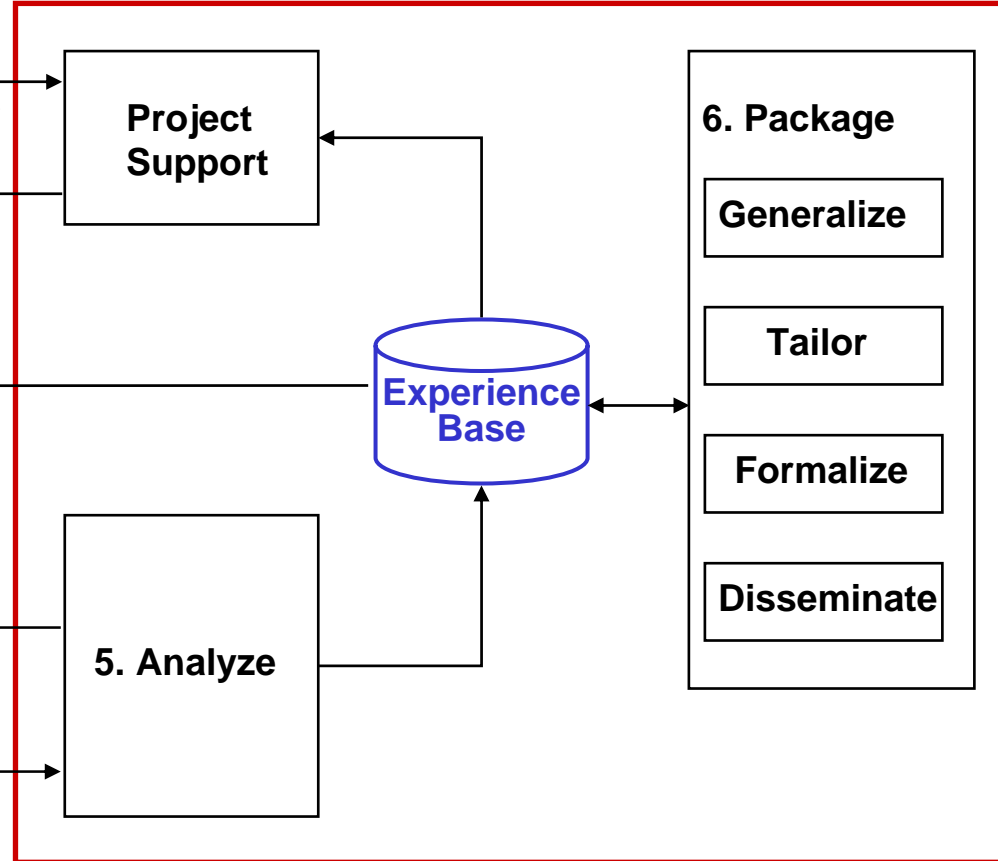


The Experience Factory Organization

Project Organization

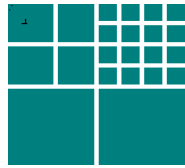


Experience Factory





The Experience Factory Organization A Different Paradigm



Project Organization Problem Solving

Experience Factory Experience Packaging

Decomposition of a problem
into simpler ones

Unification of different solutions
and re-definition of the problem

Instantiation

Generalization, Formalization

Design/Implementation process

Analysis/Synthesis process

Validation and Verification

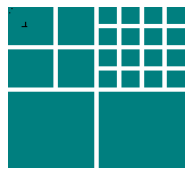
Experimentation

**Product Delivery within
Schedule and Cost**

**Experience / Recommendations
Delivery to Project**



SEL: An Example Experience Factory Structure



PO

EF

DEVELOPERS (SOURCE OF EXPERIENCE)

STAFF	275-300 developers
TYPICAL PROJECT SIZE	100-300 KSLOC
ACTIVE PROJECTS	6-10 (at any given time)
PROJECT STAFF SIZE	5-25 people
TOTAL PROJECTS (1976-1994)	120
<i>NASA + CSC</i>	

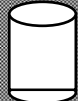


PROCESS ANALYSTS (PACKAGE EXPERIENCE FOR REUSE)

STAFF	10-15 Analysts
FUNCTION	<ul style="list-style-type: none"> • Set goals/questions/metrics - Design studies/experiments • Analysis/Research • Refine software process - Produce reports/findings
PRODUCTS (1976-1994)	300 reports/documents
<i>NASA + CSC + U of MD</i>	

Development measures for each project

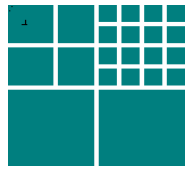
Refinements to development process

DATA BASE SUPPORT (MAINTAIN/QA EXPERIENCE INFORMATION)

STAFF	3-6 support staff		
FUNCTION	<ul style="list-style-type: none"> • Process forms/data • QA all data • Record/archive data • Maintain SEL data base • Operate SEL library 	<ul style="list-style-type: none"> SEL DATA BASE  160 MB FORMS LIBRARY  220,000 REPORTS LIBRARY  <ul style="list-style-type: none"> • SEL reports • Project documents • Reference papers 	
<i>NASA + CSC</i>			



Using Baselines to Show Improvement 1987 vs. 1991 vs. 1995



Continuous Improvement in the SEL

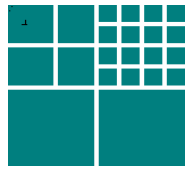
Decreased **Development Defect rates** by
 75% (87 - 91) **37%** (91 - 95)
Reduced **Cost** by
 55% (87 - 91) **42%** (91 - 95)
Improved **Reuse** by
 300% (87 - 91) **8%** (91 - 95)
Increased **Functionality** five-fold (76 - 92)

CSC officially assessed as CMM level 5 and ISO certified (1998),
starting with SEL organizational elements and activities

These successes led to

Fraunhofer Center for Experimental Software Engineering - 1997

CeBASE Center for Empirically-based Software Engineering - 2000



CeBASE

Center for Empirically Based Software Engineering

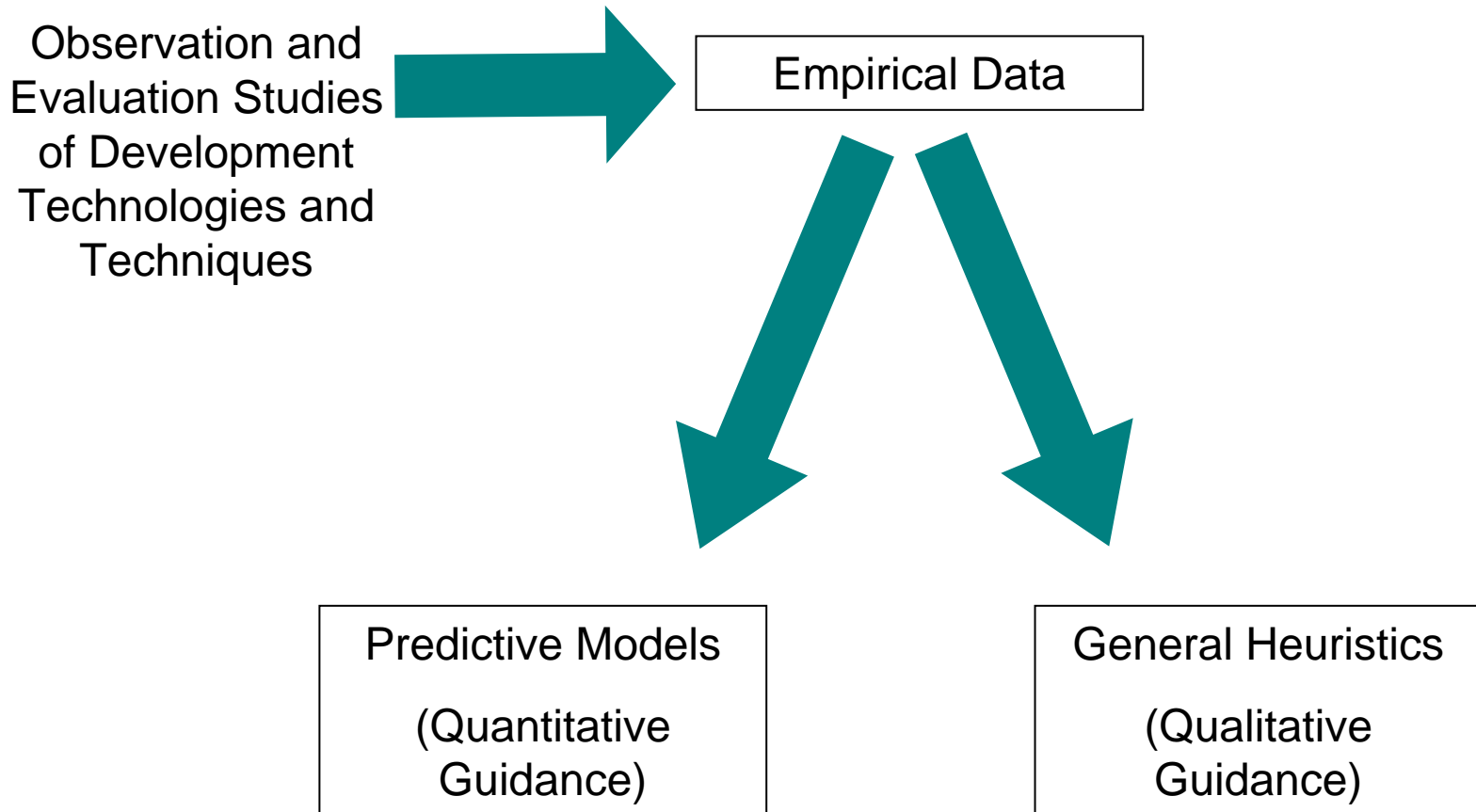
CeBASE Project Goal: Enable a **decision framework and experience base** that forms a basis and infrastructure needed to evaluate and choose among software development technologies

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: Victor Basili (UMD), Barry Boehm (USC)



CeBASE Approach

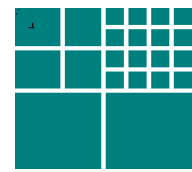


E.g. COCOTS excerpt:

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.



CeBASE

Three-Tiered Empirical Research Strategy

Technology maturity

Primary activities

Evolving results

Practical applications
(Government, industry, academia)

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

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

Applied Research

Experimentation and analysis with the concepts in selected areas.

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

Basic Research

Building a SE Empirical Research Engine and Experience base structure

Empirical methods for SE, Experience Base definition, decision support structure



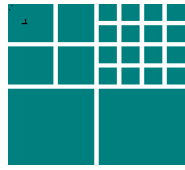


CeBASE Basic Research Activities



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

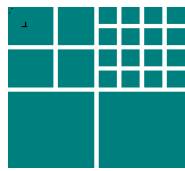
Problem: How do you elicit the software dependability needs of various stakeholders and what technologies should be applied to achieve that level of dependability?

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

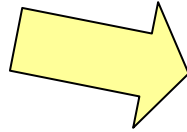
Research Goal: Quantitatively define dependability, develop high dependability technologies and assess their effectiveness under varying conditions and transfer them into practice

Partners: NASA, CMU, MIT, UMD, USC, U. Washington, Fraunhofer-MD

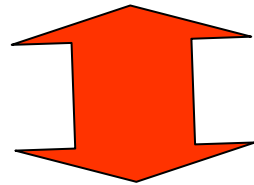
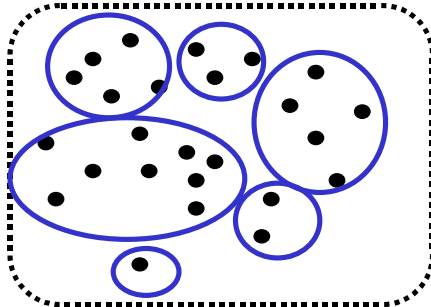
What are the top level research problems?



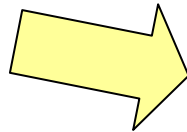
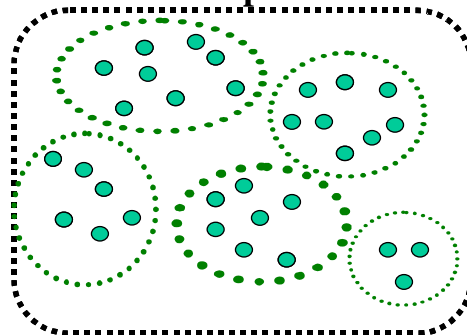
System Users



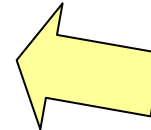
Failures Space



Fault Space



Technology Developers



System Developers

Research Problem 1

Can the quality needs be understood and modeled?

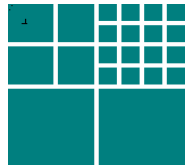
Research Problem 3

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

Research Problem 2

What does a technology do?

Can it be empirically demonstrated?



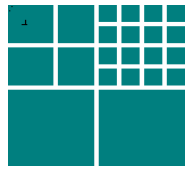
System User Issues

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

issue

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

manifest

cause

event

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

trigger

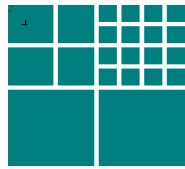
event

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

reaction

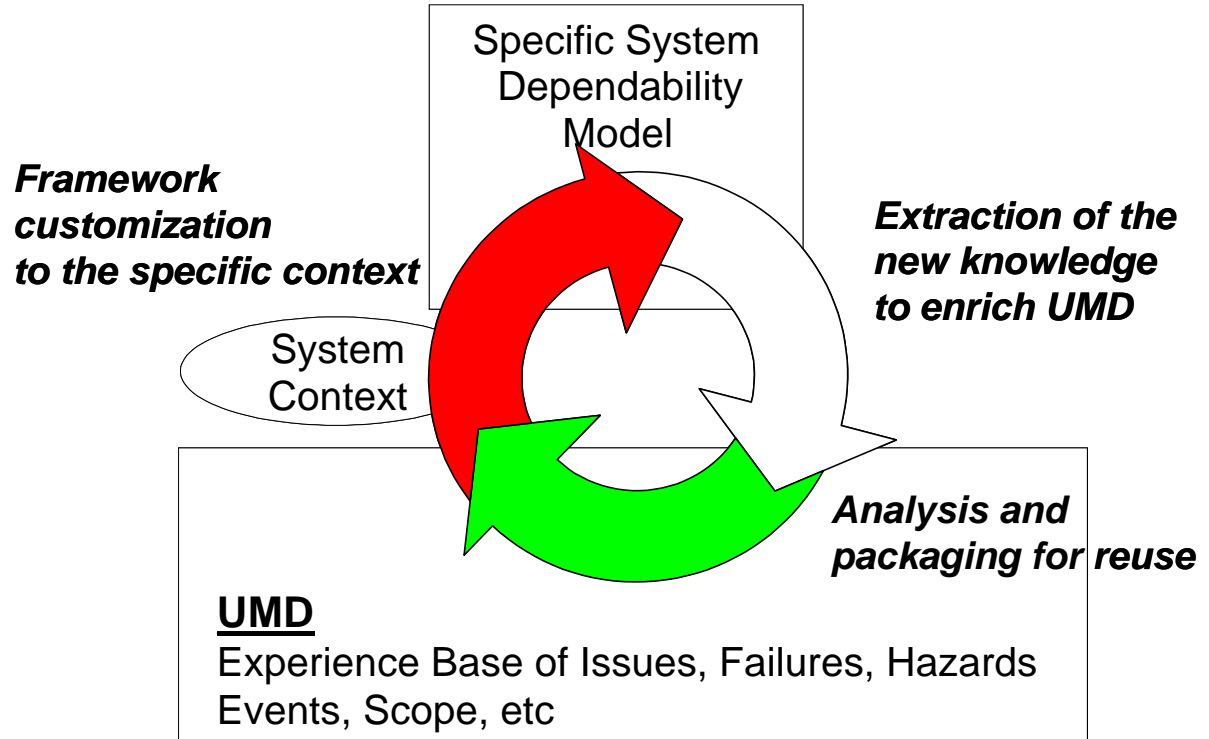
- **Impact mitigation**
- warnings
- alternative services
- mitigation services
- **Recovery**
- recovery time / actions
- **Occurrence reduction**
- guard services

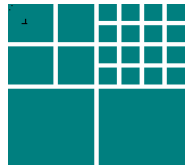
UMD assimilates new experience



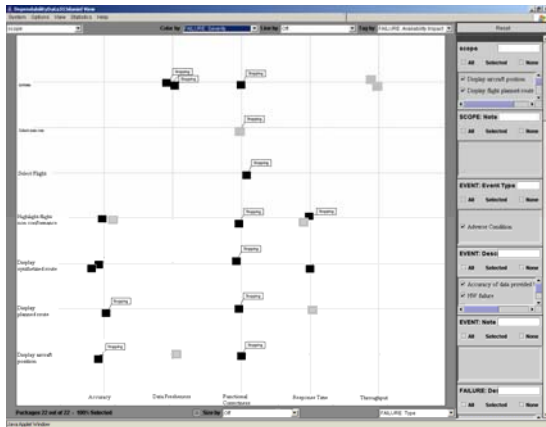
Characterizations (e.g., types, severity, etc.) of the basic UMD modeling concepts of issue, scope, measure, and event depend on the specific context (project and stakeholders).

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

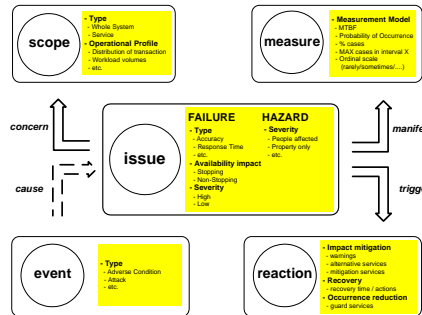
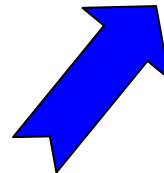
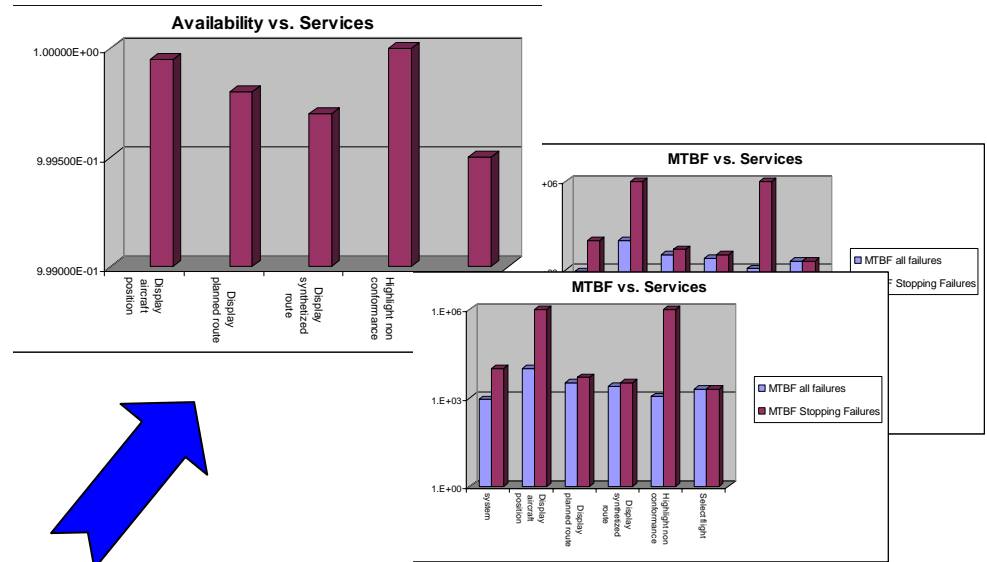
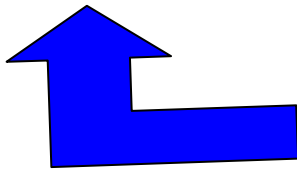




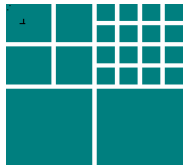
UMD support engineering decisions at requirements phase for quality validation, negotiation, trade-offs analysis



Requirements Visualization



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



Technology 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?

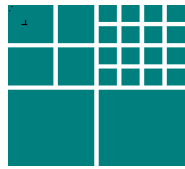
Mid-process iterations concentrated on **usability**:

- *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?

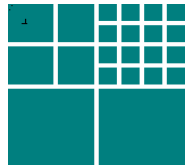
Most recent iterations concentrated on **effectiveness**:

- *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?*



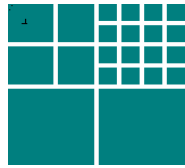
Using testbeds to transfer technology

- 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

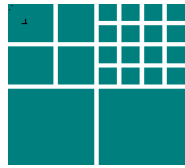
- **Testbed :**
 - 40 versions of TSAFE source code were created via fault seeding
 - The faults were created to resemble possible errors that can arise in using the concurrency controller pattern such as
 - making an error while writing a guarded command or
 - forgetting to call a concurrency controller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



Applied Research

DoE High Productivity Computing Systems

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

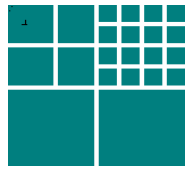
Time to Solution = Development Time + Execution Time

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.

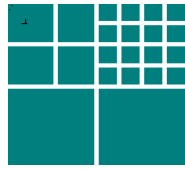
Partners: MIT Lincoln Labs, MIT, UCSD, UCSB, UMD, USC, FC-MD



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?
 - ...

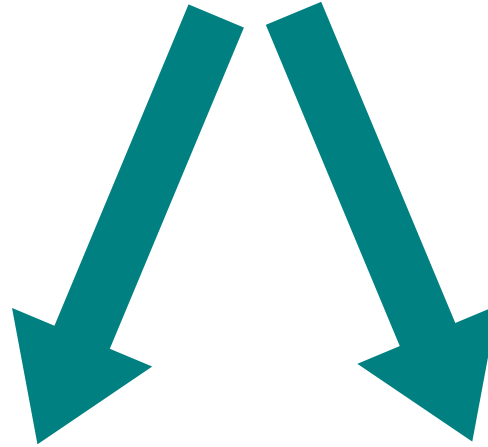


HPCS Research Activities

Development Time
Experiments –
Novices and Experts



Empirical Data



Predictive Models
(Quantitative
Guidance)

General Heuristics
(Qualitative
Guidance)

E.g. Tradeoff between effort and performance:

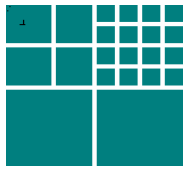
MPI will increase the development effort by $y\%$ and increase the performance $z\%$ over **OpenMP**

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:

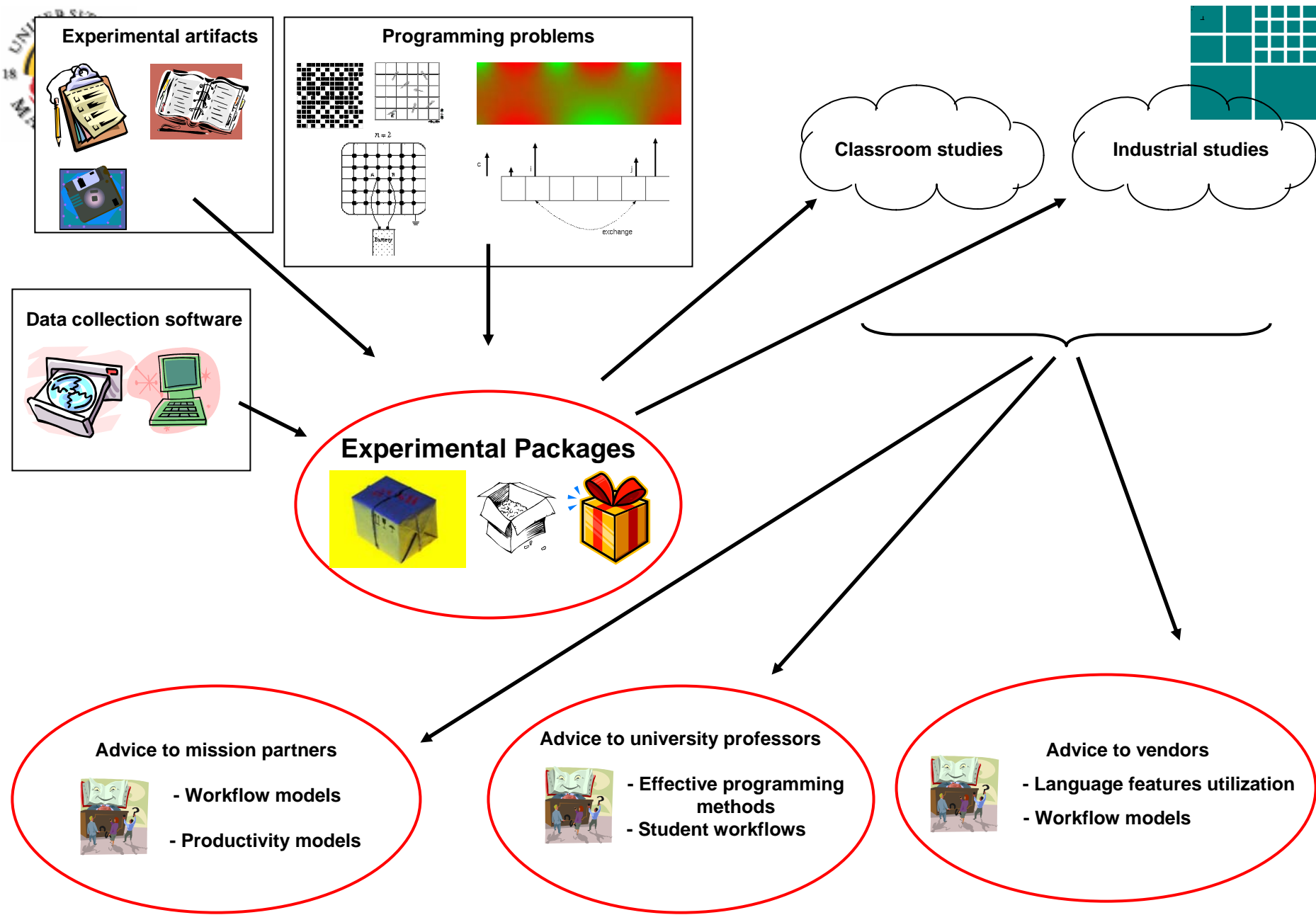
- **Classroom assignments** (Array Compaction, the Game of Life, Parallel Sorting, LU Decomposition, ...)

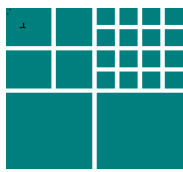
to

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

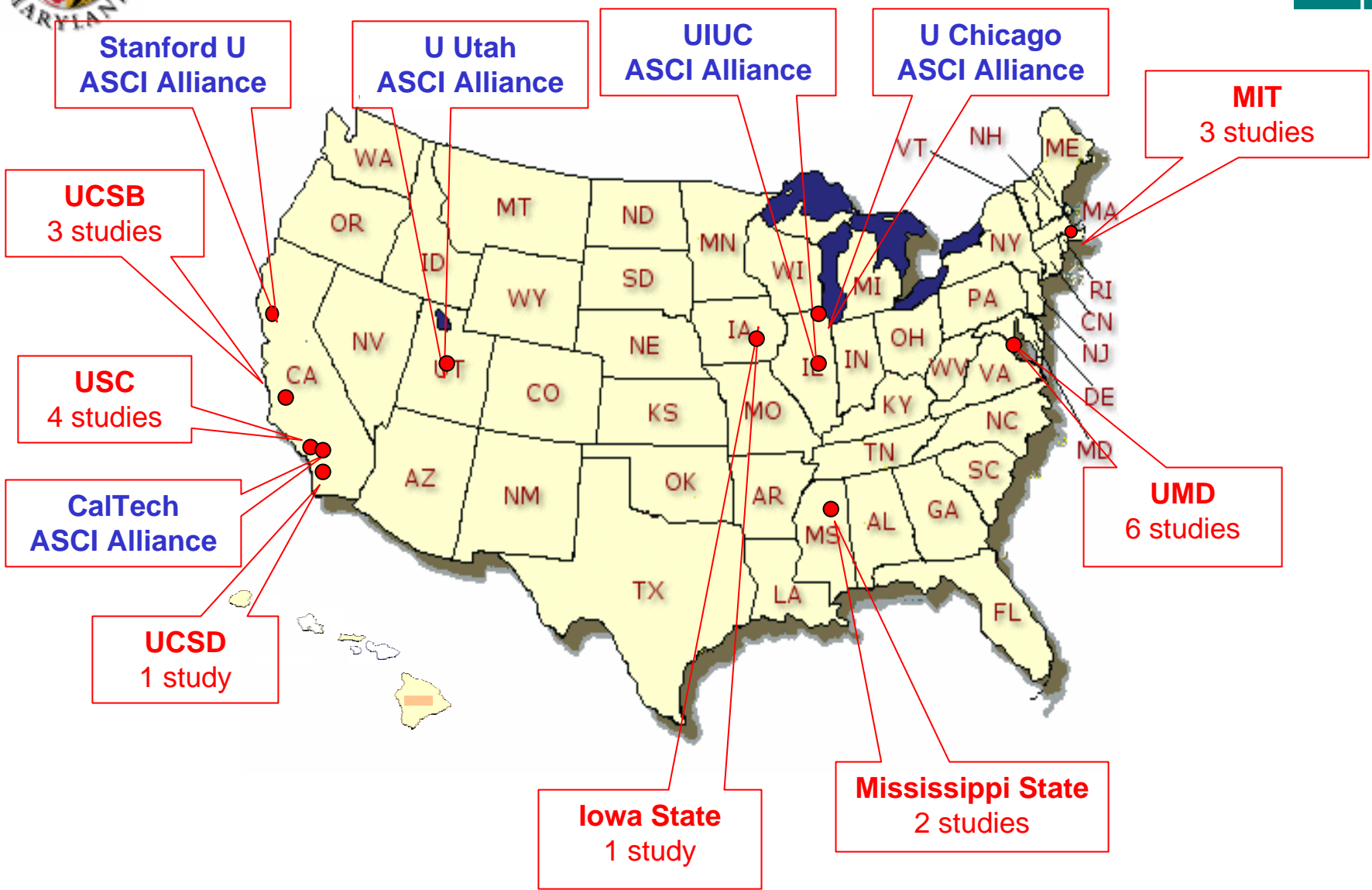
to

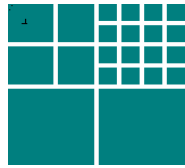
- **Full scientific applications** (nuclear simulation, climate modeling, protein folding,)





Studies Conducted





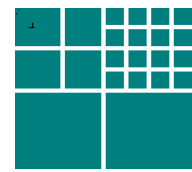
Clearinghouse Project

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

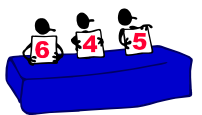
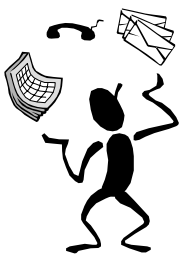
Research Goal: Define a repeatable model-based empirical evidence vetting process enabling different people to create profiles consistently and the integration of new evidence

Partners: OSD, UMD, FC-MD, DAU, CSC, ...



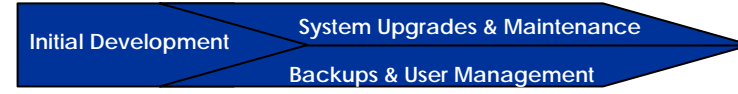
Operational Concept

Best Practice Handling

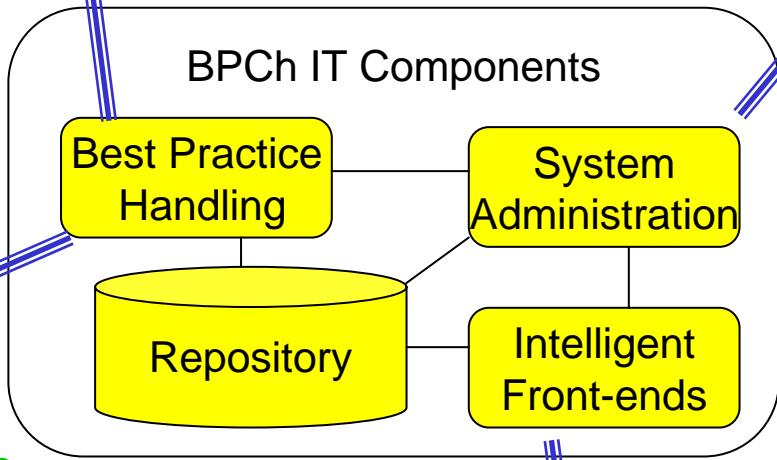


Information Handlers

BPCh Operations



Support Team



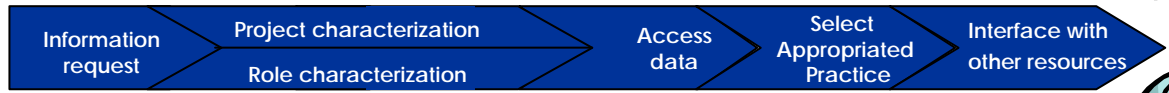
Best Practice Contributions



Information Providers



BPCh Usage



Information Seekers

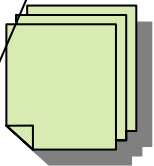


Behind the Scenes

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?



Evidence 1

Source
Context
Results



Evidence 2

Source
Context
Results



Evidence 3

Source
Context
Results



Evidence 4

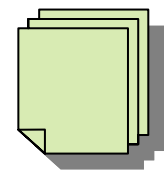
Source
Context
Results

Behind the Scenes

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



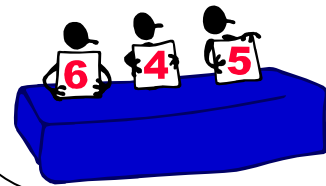
Evidence 3

Source
Context
Results



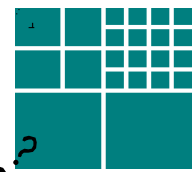
Evidence 4

Source
Context
Results





The User View



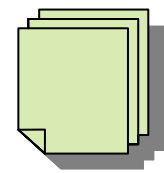
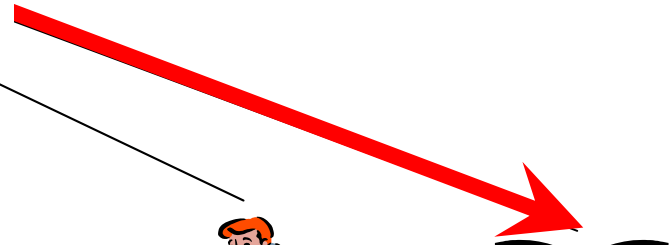
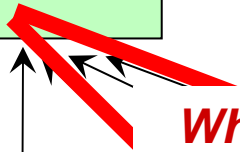
Acquisition manager, safety critical program

Help me find a practice to reduce schedule.

Summary



Who's used it for safety critical programs?



Evidence 1

Evidence 2

Evidence 3

Evidence 4

Source
Context
Results

Source
Context
Results

Source
Context
Results

Source
Context
Results



Summarizing

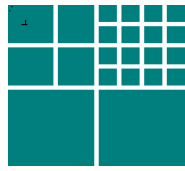


- Measurement is fundamental to any engineering science
- User needs must be made explicit (measurable models)
- Organizations have different characteristics, goals, cultures; stakeholders have different needs
- Process is a variable and needs to be selected and tailored to solve the problem at hand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