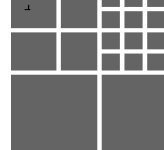


The Role of Empirical Study in Software Engineering

Victor R. Basili

**Professor, University of Maryland
and**

Director, Fraunhofer Center - Maryland

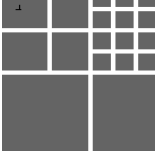


Outline

- Empirical Studies
 - Motivation
 - Specific Methods
 - Example: SEL
- Applications
 - CeBASE
 - NASA High Dependability Computing Project
 - The Future Combat Systems Project
 - DoE High Productivity Computing System



Motivation for 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



Motivation for 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

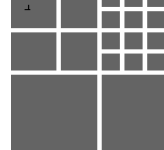
- research can provide models to help

Research and Development have a symbiotic relationship

requires a working relationship between industry and academe



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 the demonstrate sustained, measurable improvement?

More specifically:

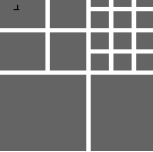
When are peer reviews more effective than functional testing?

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

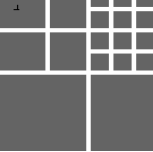
- Peer reviews are more effective than functional testing for faults of omission and incorrect specification (UMD, USC)
- Functional testing is more effective than reviews for faults concerning numerical approximations and control flow (UMD, USC)

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)

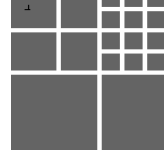


Basic Concepts for Empirical Software Engineering



This process of model building, experimentation and learning requires the development, tailoring and evolution of methods that support evolutionary learning,
closed loop processes,
well established measurement processes and
the opportunity to build software core competencies

As well as processes that support the development of software that is relevant to the needs of the organization
can be predicted and estimated effectively
satisfies all the stakeholders
does not contain contradictory requirements



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

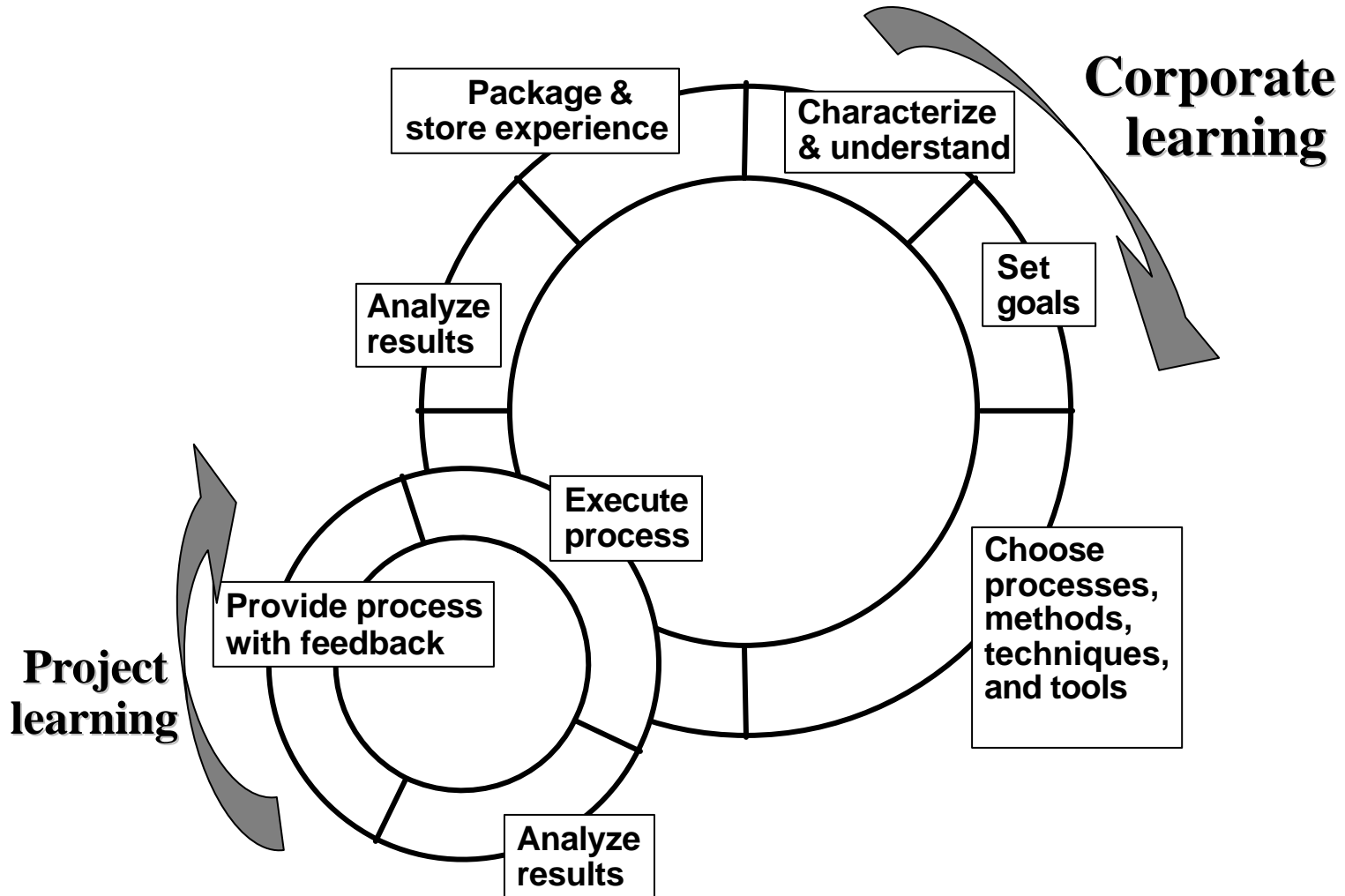
Goal/Question/Metric Paradigm (GQM)

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

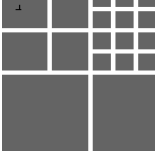
Experience Factory (EF)

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

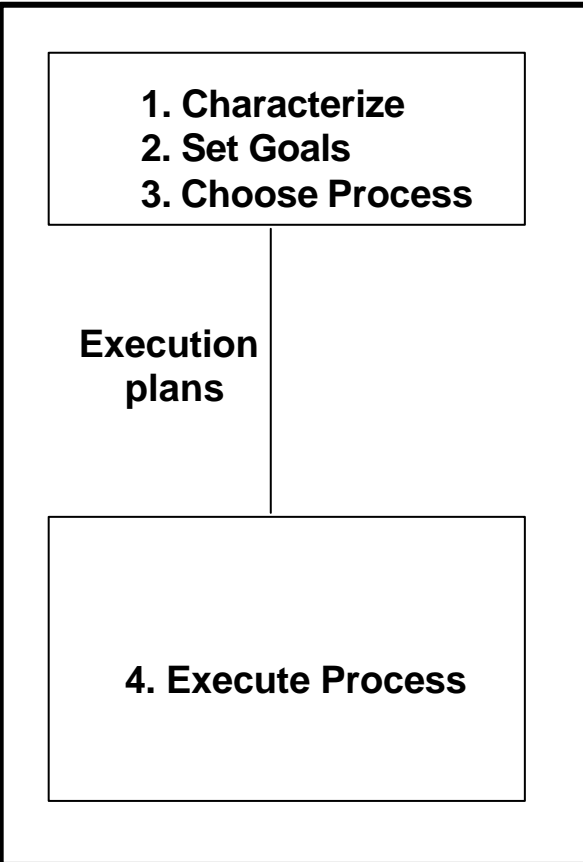
Quality Improvement Paradigm



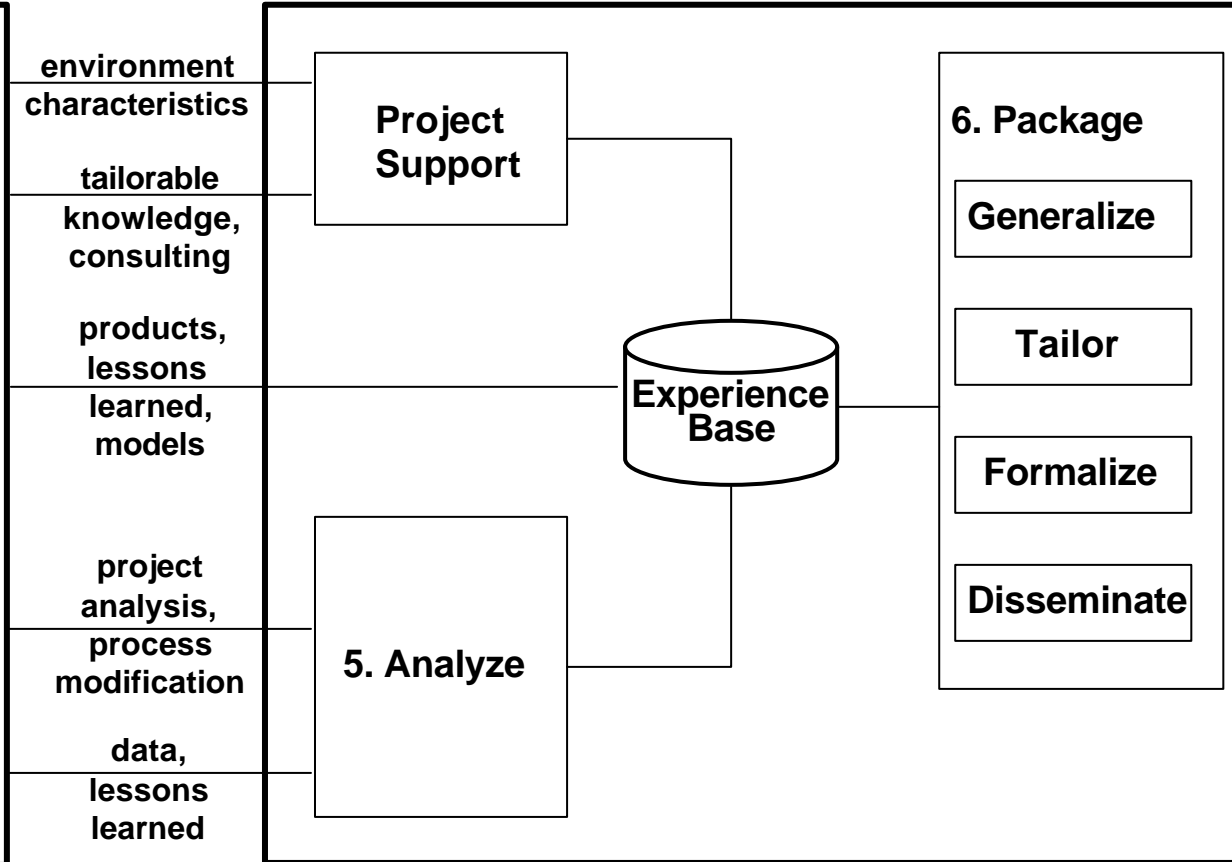
The Experience Factory Organization



Project Organization



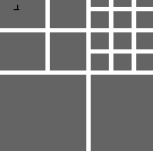
Experience Factory





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

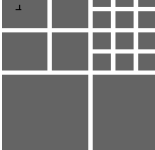
Experimentation

**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>	

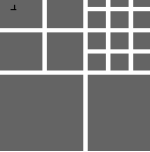
Development measures for each project →

← Refinements to development process

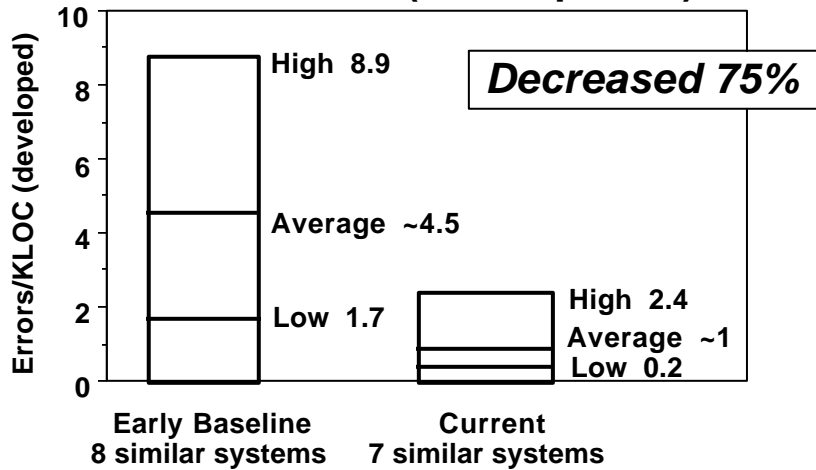
DATA BASE SUPPORT (MAINTAIN/QA EXPERIENCE INFORMATION)

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 		
		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>			

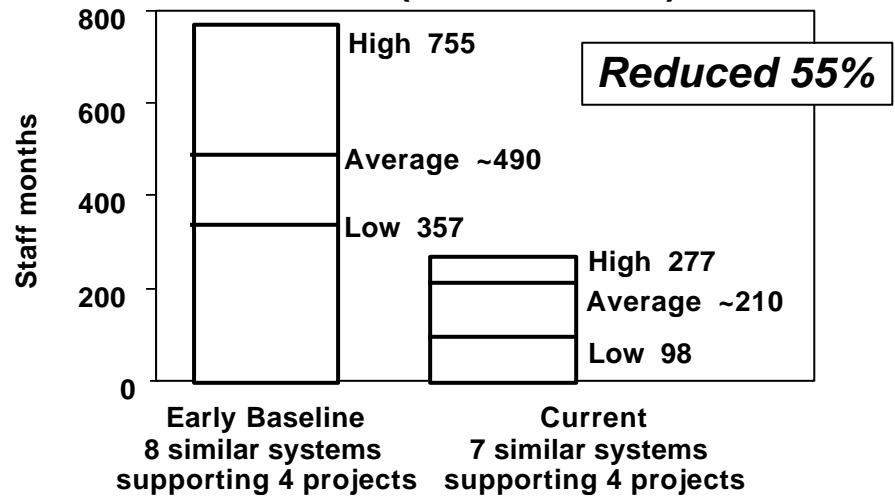
Using Baselines to Show Improvement 1987 vs. 1991



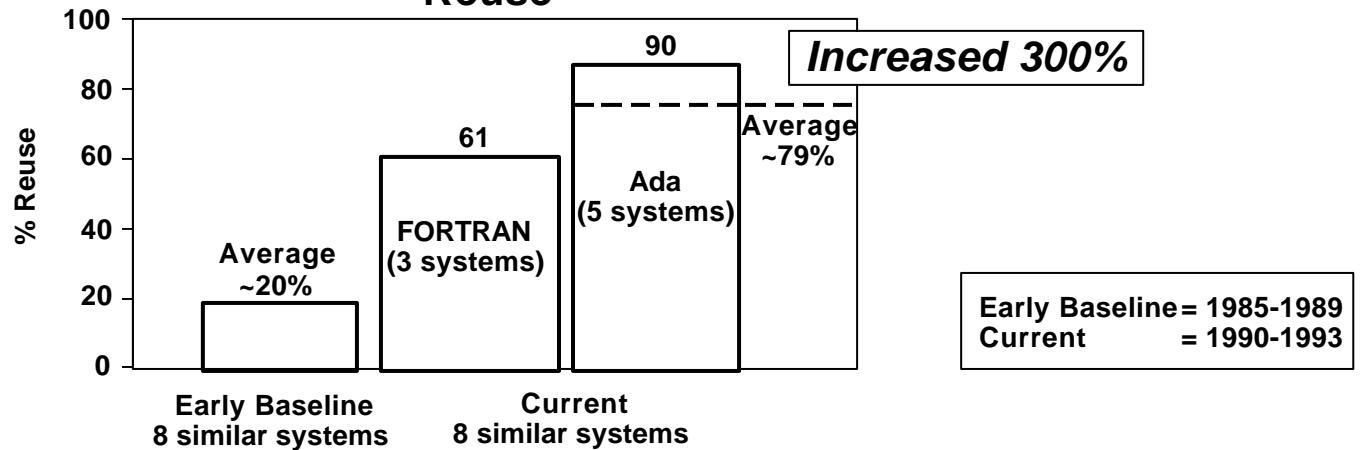
Error Rates (development)



Cost (staff months)

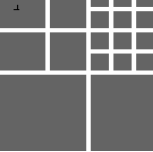


Reuse





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)

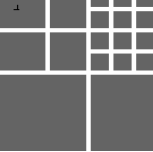
CSC officially assessed as CMM level 5 and ISO certified (1998),
starting with SEL organizational elements and activities

Fraunhofer Center for Experimental Software Engineering - 1998

CeBASE Center for Empirically-based Software Engineering - 2000



Empirical Software Engineering Needs



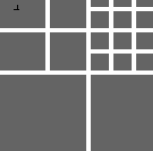
Interact with various industrial, government and academic organizations to open up the domain for learning

Partner with other organizations to expand the potential competencies

Observe and gather as much information as possible

Analyze and synthesize what has been learned into sets of best practices recognizing what has been effective and under what circumstances allowing for tailoring based up context variables

Package results for use and feed back what has been learned to improve the practices



Example: CeBASE

Center for Empirically Based Software Engineering

The **CeBASE** project was created to support the symbiotic relationship between research and development, academia and industry

Virtual Research Center

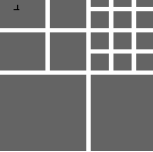
Created by the NSF Information Technology Research Program

Co-Directors: Victor Basili (UMD), Barry Boehm (USC)

Initial technology focus: Defect reduction techniques, COTS based development, Agile Methods

CeBASE Framework

Experience Factory, Goal/Question/Metric Approach, Spiral Model extensions, MBASE, WinWin Negotiations, Electronic Process Guide, eWorkshop collaboration, COCOMO cost family, EMS Experience Base, VQI (Virtual Query Interface)



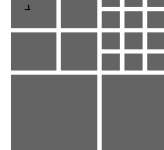
CeBASE

Center for Empirically Based Software Engineering

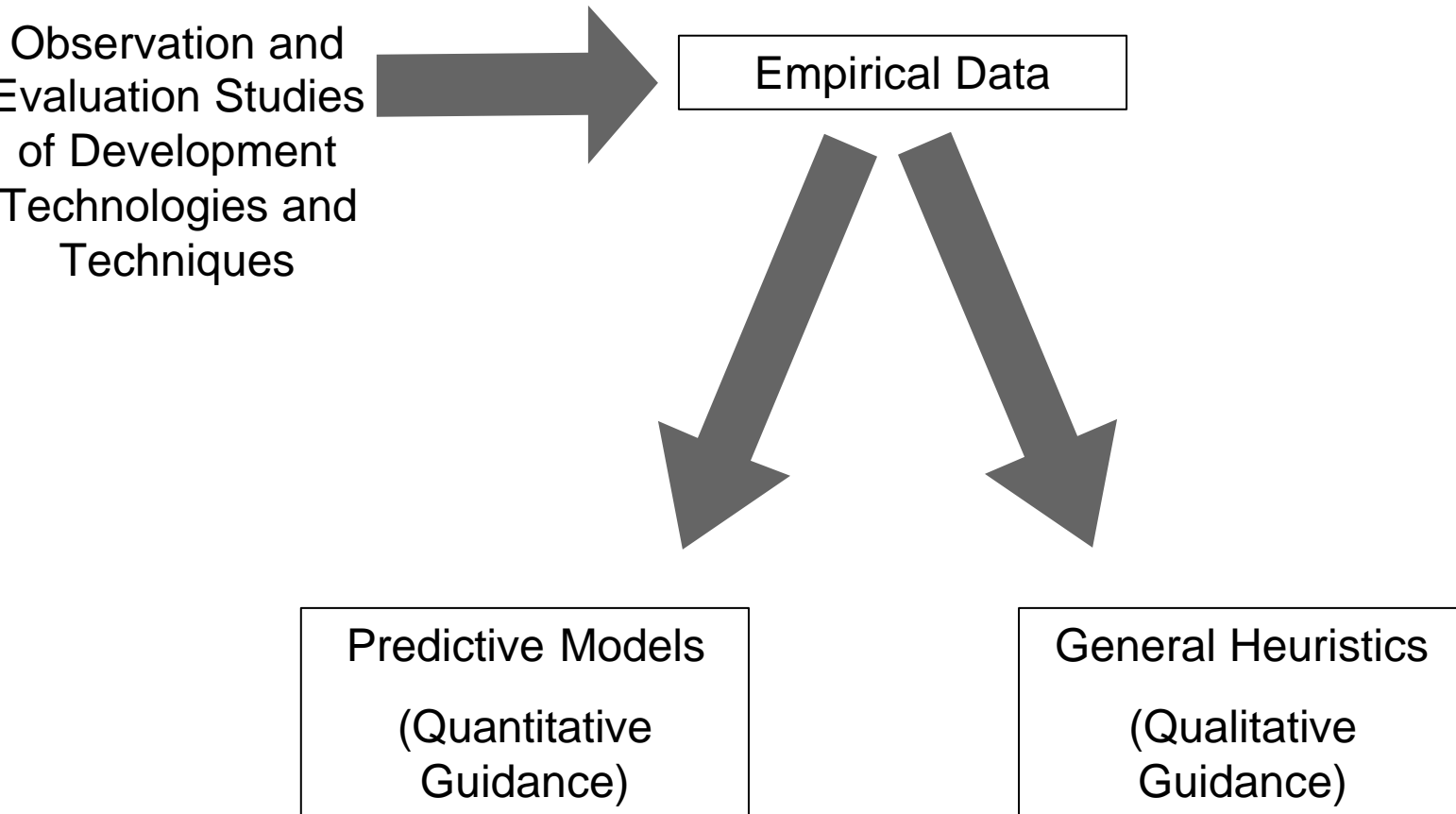
CeBASE Project Goal: Enable a **decision framework and experience base** that forms a basis and an infrastructure for research and education in empirical methods and software engineering

CeBASE Research Goal: Create and evolve an **empirical research engine** for evaluating and choosing among software development technologies





CeBASE Approach



E.g. COCOTS excerpt:

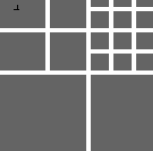
Cost of COTS tailoring = f(# parameters initialized, complexity of script writing, security/access requirements, ...)

E.g. Defect Reduction Heuristic:

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

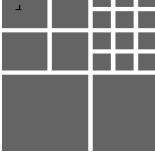


CeBASE Basic Research Activities



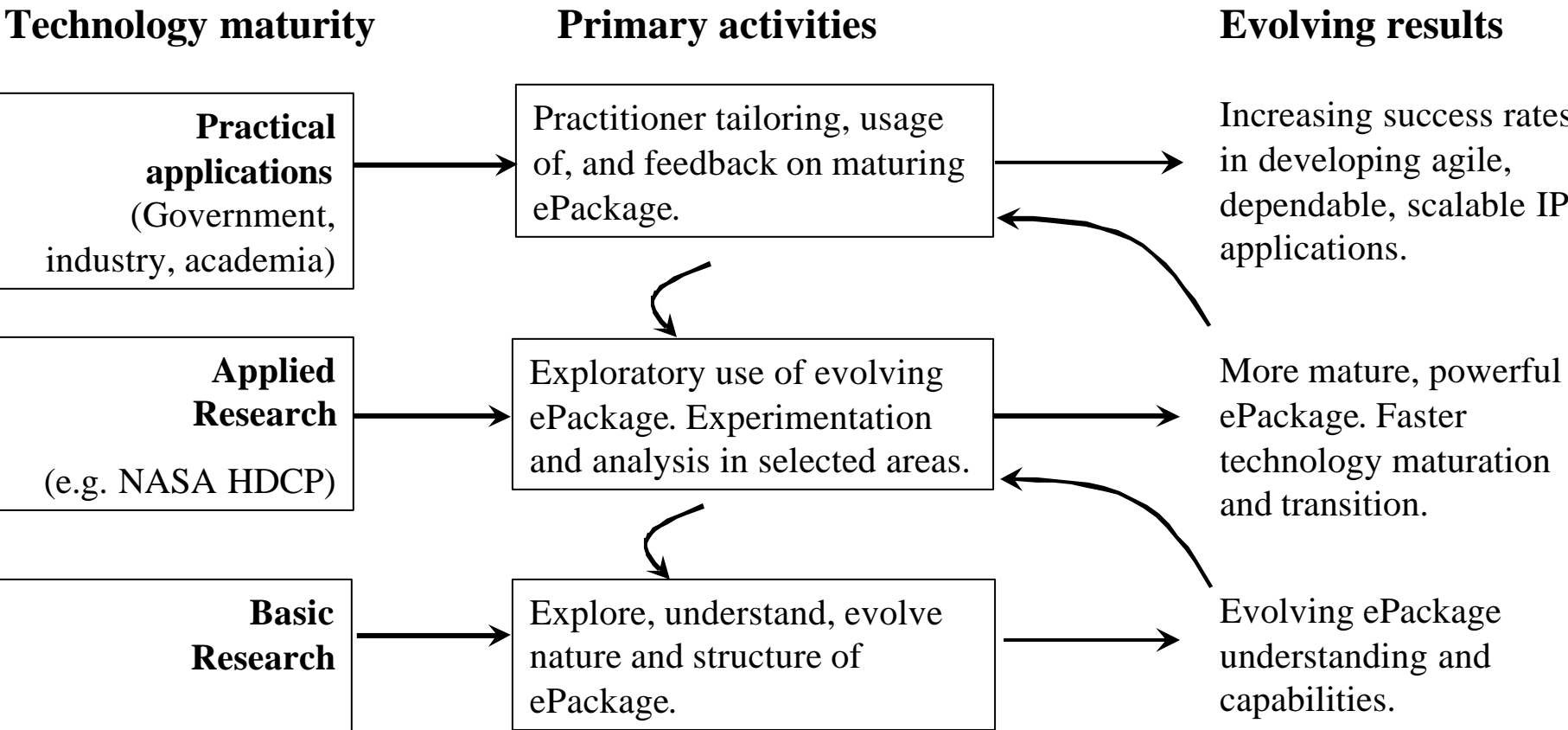
Define and improve methods to

- 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

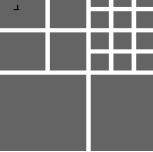


CeBASE

Three-Tiered Empirical Research Strategy



(ePackage = Empirical Research Engine, eBase, empirical decision framework)



Applied Research

NASA High Dependability Computing Program

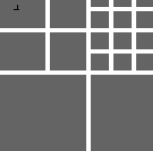
Project Goal: Increase the ability of NASA to engineer highly dependable software systems via the development of new techniques and technologies

Research Goal: Develop high dependability technologies and assess their effectiveness under varying conditions and transfer them into practice

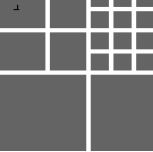
Partners: NASA, CMU, MIT, UMD, USC, U. Washington, Fraunhofer-MD



HDCCP Research Questions



- **System User**
 - How can the dependability needs be understood and modeled?
 - Elicit and operationalize stakeholders' dependability needs
- **Technology Developer**
 - What does a technology do? Can it be empirically demonstrated?
 - Formalize technology claims, seed faults in test beds, apply technologies, evaluate claim
- **System Developer**
 - How well does a set of interventions cover the system developer's "problem space"?
 - Characterize the fault classes for the organization and domain, and identify overlapping contributions
- **System Developer**
 - What set of interventions should be applied to achieve the desired dependability?
 - Matching Failures to Faults



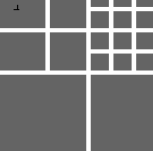
HDCP System User Issues

**How do I elicit dependability 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?



HDCP System Developer Issues

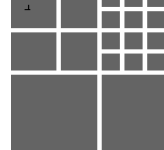


**How can I understand the stakeholders dependability needs?
How can I apply the available techniques to deliver the
required dependability?**

- 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 you identify the appropriate combinations of technologies for the project needs?
 - Technologies available, characterization, combinations of technologies to achieve goals
- How do you tailor the technologies for the project?

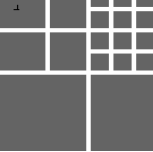


HDCP Technology Researcher Issues



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?



HDCP : Example Outcome

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**:

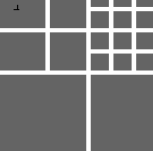
- *usability problems, results due to individual steps in the context of small systems in actual development.*

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.*

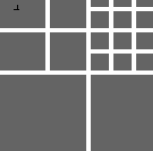
Does the new techniques represent a usable improvement to practice?



HDCP

Using testbeds to transfer technology

- **Define Testbeds**
 - Projects, operational scenarios, detailed evaluation criteria representative of NASA 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
 - provide insights into the models of dependability
- **Conduct empirical evaluations** of emerging HDCP technology
 - Establish evaluation support capabilities: instrumentation, seeded defect base; experimentation guidelines

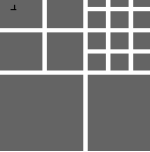


HDCP

Increasing the relevance of the testbeds

View each technology as passing through a *series* of milestones

- **M1. Internal:** Initial set of examples that the technology researcher has already developed in the research process
- **M2. Packaged domain-specific:** Set of toy examples with high dependability needs, packaged for use by the technologists, e.g. TSAFE, SCROver
- **M3. NASA off-line:** Part or all of a system previously developed for NASA, e.g., CTAS, EOSDIS
- **M4. Live examples:** Part or all of a system currently under development, e.g., MSL



Applied Research

DoE 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

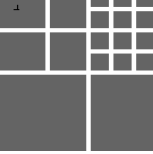
Time to Solution = Development Time + Execution Time

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



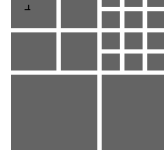
HPCS Example Questions



- How does an HPC environment (hardware, software, human) affect the development of an HPC program?
 - What is the **cost** and **benefit** of applying a particular HPC technology?
 - What are the **relationships** among the technology, the work flows, development cost 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?
 - ...



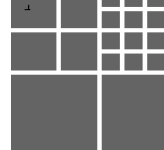
HPCS Example Hypotheses



- Effort to parallelize serial code is greater than effort to develop serial code
- Novices can achieve speedup
- The variation in execution time of MPI codes will be greater than the variation in execution time of OpenMP codes
- The variation in the speedup of MPI codes will increase with the number of processors
- ...



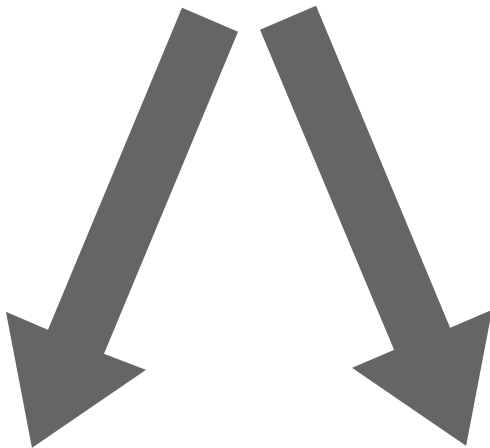
HPCS Research Activities



Development Time
Experiments –
Novices and Experts



Empirical Data



Predictive Models
(Quantitative
Guidance)

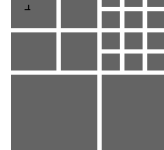
General Heuristics
(Qualitative
Guidance)

E.g. Tradeoff between effort and performance:

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

E.g. Scalability:

If you need high scalability, choose **MPI** over **OpenMP**



HPCS Testbeds

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

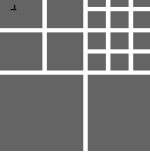
- Classroom assignments (Array Compaction, the Game of Life, Parallel Sorting, LU Decomposition, ...)

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,)

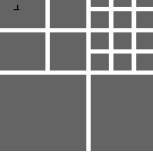


Technology Transfer Future Combat Systems

Project Goal: Support FCS Program Management Office in the development of the Future Combat Systems (FCS), focusing on the complex system of systems (software) development risk, e.g., acquisition, architecture, ... and build lessons learned for future iterations of FCS and future CSoS.

Research Goal: Build a risk experience Base and a Complex System of Systems Lessons Learned Experience Base.

Partners: UMD, USC, FC-MD, SEI, Sandia, LSI: Boeing, SAIC



FCS Technology Transfer

Assumption: the technologies are mature enough and have been shown successful in other projects or organizations

Example technologies being transferred:

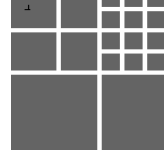
GQM to help define goals of various levels of project management for complex systems of systems

Spiral life cycle model to the development of the system

Experience base tracking problems associated with a complex system of systems to learn from early spirals of development and provide an experience base for future systems

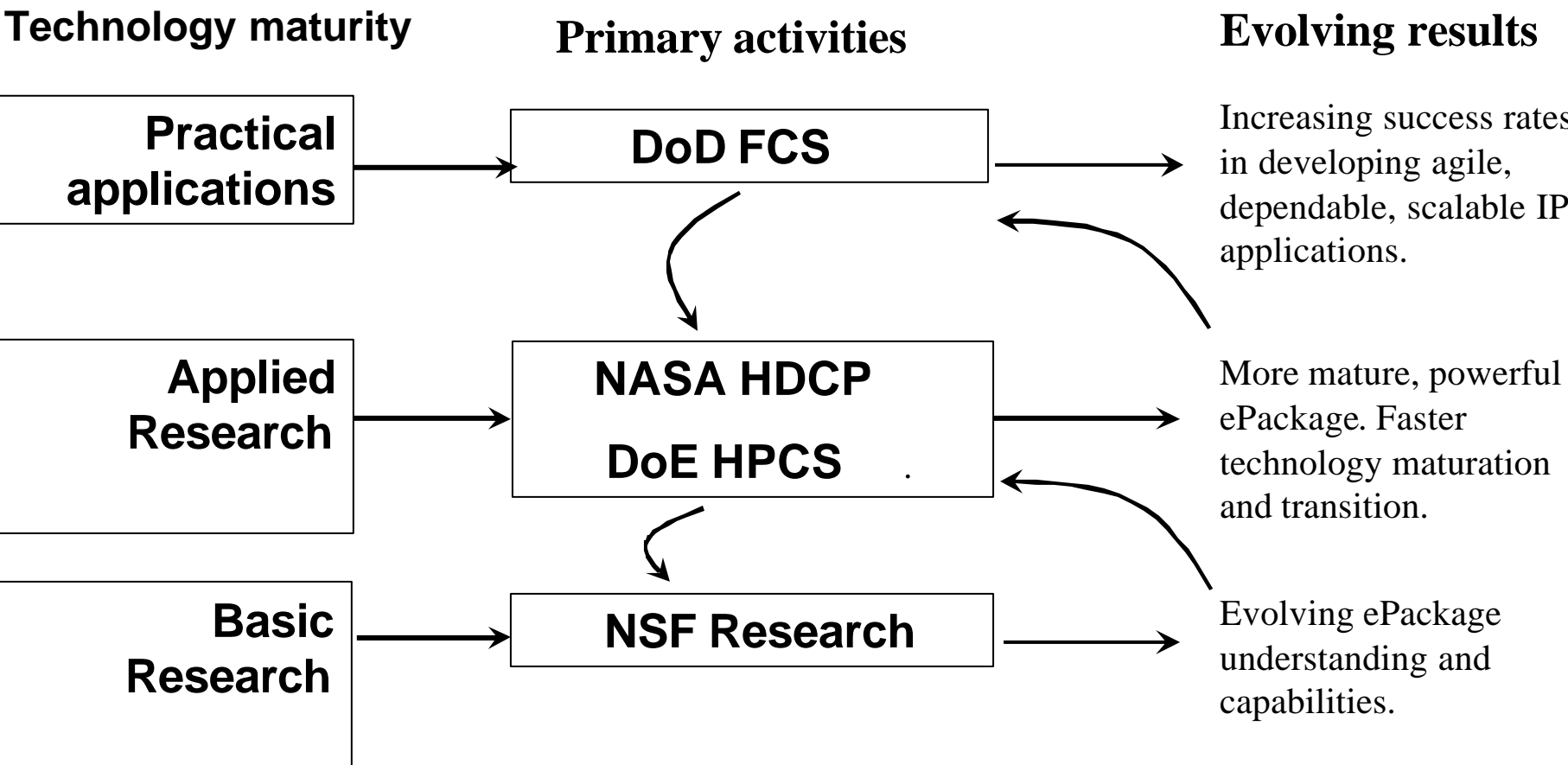
Activities: Observe, interview, tailor, train, support, learn, ...

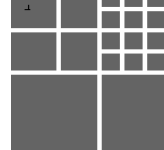
Feedback: Take what has been learned and feed it back to identify research needs, immaturity in technologies, the importance of context variables, ...



CeBASE

Three-Tiered Empirical Research Strategy





Conclusion

- This talk is about
 - The role of empirical study in software engineering
 - The **synergistic relationship between research, applied research, and practice**
- 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