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 processes 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

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

Corporate learning

- Package & store experience
- Characterize & understand
- Set goals
- Choose processes, methods, techniques, and tools
- Execute process
- Analyze results
- Provide process with feedback

Project learning
The Experience Factory Organization

**Project Organization**

1. Characterize
2. Set Goals
3. Choose Process

Execution plans

4. Execute Process

**Experience Factory**

- **Experience Base**
  - Project Support
  - 5. Analyze
    - environment characteristics
    - tailorable knowledge, consulting
    - products, lessons learned, models
    - project analysis, process modification
    - data, lessons learned

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

## A Different Paradigm

<table>
<thead>
<tr>
<th>Project Organization</th>
<th>Experience Factory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Problem Solving</strong></td>
<td><strong>Experience Packaging</strong></td>
</tr>
<tr>
<td>Decomposition of a problem into simpler ones</td>
<td>Unification of different solutions and re-definition of the problem</td>
</tr>
<tr>
<td>Instantiation</td>
<td>Generalization, Formalization</td>
</tr>
<tr>
<td>Design/Implementation process</td>
<td>Analysis/Synthesis process</td>
</tr>
<tr>
<td>Validation and Verification</td>
<td>Experimentation</td>
</tr>
<tr>
<td><strong>Product Delivery within Schedule and Cost</strong></td>
<td><strong>Experience / Recommendations Delivery to Project</strong></td>
</tr>
</tbody>
</table>
SEL: An Example Experience Factory Structure

**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

**Process Analysts** (Package Experience for Reuse)
- **Staff**: 10-15 Analysts
- **Function**
  - Set goals/questions/metrics
  - Design studies/experiments
  - Analysis/Research
  - Refine software process
  - Produce reports/findings
- **Products (1976-1994)**: 300 reports/documents

**Data Base Support** (Maintain/QA Experience Information)
- **Staff**: 3-6 support staff
- **Function**
  - Process forms/data
  - QA all data
  - Record/archive data
  - Maintain SEL database
  - Operate SEL library
- **SEL Database**: 160 MB
- **Forms Library**: 220,000
- **Reports Library**: SEL reports, Project documents, Reference papers

**Staff** and **function** categories for each group are as follows:

- SEL: An Example Experience Factory Structure

**Diagram**

- **PO**
- **Developers**
- **Process Analysts**
- **Data Base Support**
- **NASA + CSC + U of MD**

**Additional Notes**

- **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

- **Function**
  - Set goals/questions/metrics
  - Design studies/experiments
  - Analysis/Research
  - Refine software process
  - Produce reports/findings

- **Products (1976-1994)**: 300 reports/documents

- **Staff**: 3-6 support staff
- **Function**
  - Process forms/data
  - QA all data
  - Record/archive data
  - Maintain SEL database
  - Operate SEL library

- **NASA + CSC + U of MD**
Using Baselines to Show Improvement
1987 vs. 1991

Error Rates (development)
- Early Baseline: 8 similar systems
  - High: 8.9
  - Average: ~4.5
  - Low: 1.7
- Current: 7 similar systems
  - High: 2.4
  - Average: ~1
  - Low: 0.2

Decreased 75%

Cost (staff months)
- Early Baseline: 8 similar systems supporting 4 projects
  - High: 755
  - Average: ~490
  - Low: 357
- Current: 7 similar systems supporting 4 projects
  - High: 277
  - Average: ~210
  - Low: 98

Reduced 55%

Reuse
- Early Baseline: 8 similar systems
  - FORTRAN (3 systems): Average ~20%
  - ADA (5 systems): Average ~79%
- Current: 8 similar systems
  - 61
  - 90
  - Increased 300%

IEEE59
Early Baseline = 1985-1989
Current = 1990-1993
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  
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

Observation and Evaluation Studies of Development Technologies and Techniques

Empirical Data

Predictive Models
(Quantitative Guidance)

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

General Heuristics
(Qualitative Guidance)

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

Technology maturity

- **Basic Research**
  - **Basic Research**

- **Applied Research**
  - **Applied Research** (e.g. NASA HDCP)

- **Practical applications**
  - **Practical applications** (Government, industry, academia)

Primary activities

- **Practitioner tailoring, usage of, and feedback on maturing ePackage.**

- **Exploratory use of evolving ePackage. Experimentation and analysis in selected areas.**

- **Explore, understand, evolve nature and structure of ePackage.**

Evolving results

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

- More mature, powerful ePackage. Faster technology maturation and transition.

- Evolving ePackage understanding and capabilities.

(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
HDCP 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

\[
\text{Time to Solution} = \text{Development Time} + \text{Execution Time}
\]

**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)

E.g. Tradeoff between effort and performance:

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

General Heuristics
(Qualitative Guidance)

E.g. 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
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

Technology maturity

- Practical applications
- Applied Research
- Basic Research

Primary activities

- DoD FCS
- NASA HDCP
- DoE HPCS
- NSF Research

Evolving results

- Increasing success rates in developing agile, dependable, scalable IP applications.
- More mature, powerful ePackage. Faster technology maturation and transition.
- Evolving ePackage understanding and capabilities.
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