
University of Maryland
Fraunhofer Center for Empirical Software Engineering

basili@cs.umd.edu
Evolving the Software Engineering Discipline
Model Building, Experimenting, and Learning

Outline

Evolving Software Engineering knowledge through experimentation

Nature of the Software Engineering Discipline

Evolution of the Discipline: What we have learned over 40 + years

Where do we have to go?
Evolving Knowledge
Model Building, Experimenting, and Learning

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

Checking our understanding is correct involves

- testing our models
- **experimentation**

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

The understanding of a discipline **evolves** over time.

Knowledge encapsulation allows us to deal with higher levels of **abstraction**.

This is the paradigm that has been used in **many fields**, e.g., physics, medicine, manufacturing.
Evolving Knowledge
Model Building, Experimenting, and Learning

What do these fields have in common?

They evolved as disciplines when they began applying the cycle of model building, experimenting, and learning

Began with observation and the recording of what was observed

Evolved to manipulating the variables and studying the effects of change in the variables

What are the differences of these fields?

Differences are in the objects they study, the properties of those object, the properties of the system that contain them, the relationship of the object to the system, and the culture of the discipline

This effects

how the models are built
how the experimentation gets done
Evolving Knowledge
Model Building, Experimenting, and Learning

Physics
- understand and predict the behavior of the physical universe
- researchers: theorists and experimentalists
- has progressed because of the interplay between the groups

Theorists build models to explain the universe
- predict the results of events that can be measured
- models based on
  - theory about the essential variables and their interaction
  - data from prior experiments

Experimentalists observe, measure, experiment to
- test or disprove a hypothesis or theory
- explore a new domain

But at whatever point the cycle is entered there is a modeling, experimenting, learning and remodeling pattern

Early experimentalists only observed, did not manipulate the objects
Modern physicists have learned to manipulate the physical universe, e.g. particle physicists.
Evolving Knowledge
Model Building, Experimenting, and Learning

Medicine
- researcher and practitioner
- clear relationship between the two
- knowledge built by feedback from practitioner to researcher

Researcher aims at understanding the workings of the human body to predict effects of various procedures and drugs

Practitioner applies knowledge by manipulating processes on the body for the purpose of curing it

Medicine began as an art form
- evolved as a field when it began observation and model building

Experimentation
- from controlled experiments to case studies
- human variance causes problems in interpreting results
- data may be hard to acquire

However, our knowledge of the human body has evolved over time
Evolving Knowledge
Model Building, Experimenting, and Learning

Manufacturing
- domain researcher and manufacturing researcher
- understand the process and the product characteristics
- produce a product to meet a set of specifications

Manufacturing evolved as a discipline when it began process improvement

Relationship between process and product characteristics
- well understood

Process improvement based upon models of
- problem domain and solution space
- evolutionary paradigm of model building, experimenting, and learning
- relationship between the three

Models are built with good predictive capabilities
- same product generated, over and over, same processes
- understanding of relationship between process and product
Like other disciplines, software engineering requires the cycle of model building, experimentation, and learning.

The study of software engineering is a laboratory science.

The researcher’s role is to understand the nature of the processes, products and their relationship in the context of the system.

The practitioner’s role is to build “improved” systems, using the knowledge available.

More than the other disciplines these roles are symbiotic.

The researcher needs laboratories to observe and manipulate the variables - they only exist where practitioners build software systems.

The practitioner needs to better understand how to build better systems - the researcher can provide models to help.
Software Engineering
The Nature of the Discipline

Software engineering is development not production

The technologies of the discipline are human based

All software is not the same
- there are a large number of variables that cause differences
- their effects need to be understood

Currently,
- insufficient set of models that allow us to reason about the discipline
- lack of recognition of the limits of technologies for certain contexts
- there is insufficient analysis and experimentation
Evolution of the ESE Discipline

• Phase I (~1970s)
  – Isolated studies

• Phase II (~1980s)
  – Multiple Studies in one domain

• Phase III (~1990s)
  – Tying Studies Together

• Phase IV (~2000s)
  – Expanding Studies Across Domains and Environments

• Phase V (Now and the future)
Phase I: ~ 1970s
Running isolated studies for a particular purpose

Kinds of Questions:

Can we quantitatively measure the effect of the application of a particular approach on the product?

Can we distinguish products developed with different methods and techniques?
Phase I
Isolated Studies for a Particular Purpose

- **Situation:**
  - Minimum amount of guidance for experimentation in the Software Engineering domain
  - Minimum opportunities for publication
  - Hard to convince the community this was important
  - ...

- **Learned:**
  - About running controlled experiments
  - Using nonparametric statistics
  - Limits of automated measures,
  - ....
Phase II: ~1980s
Multiple Studies in one Environment & Domain

Kinds of Questions:

Can we use other organization’s models as they are?

Can we build baselines of various project variables (defects, effort) and identify where methods might make a difference?

Can we collect, analyze and store large amounts of project data and feedback information to the project and the organization?

Can we increase the quality of a particular class of systems over time by observing and identifying opportunities for improvement?
Phase II
Multiple Studies in one Environment & Domain

• **Situation:**
  – Had the opportunity to learn through longitudinal studies
  – A small community began to take such studies seriously
  – …

• **Learned:**
  – Data collection needs to be goal driven
  – A measurable relationship between process and product
  – About applying the scientific method to the software domain
  – Importance of understanding the environment (context)
  – A variety of experiences can be reused, e.g., process, product, resource, defect and quality models within the environment
  – All experience needs to be evaluated, tailored, packaged for reuse in a variety of ways, and integrated
Personal Example
Observation, Feedback, Learning, Packaging

Software Engineering Laboratory (SEL)


Goals have been to
- better understand software development
- improve the process and product quality for Ground Support Software

Used the SEL as a laboratory to build models, test hypotheses,
Used the University to test high risk ideas
Developed technologies, methods and theories when necessary
Learned what worked and didn’t work, applied ideas when applicable
Kept the business going with an aim at improvement, learning
Quality Improvement Paradigm

**Characterize** the current project and its environment with respect to the appropriate models and metrics

**Set** quantifiable **goals** for project and corporate success and improvement

**Choose** the appropriate project **processes**, supporting methods and tools

**Execute** the **processes**, construct the products, collect, validate and analyze the data to provide real-time feedback for corrective action

**Analyze** the **data** to evaluate current practices, determine problems, record findings, recommend improvements for future project

**Package** the **experience** in the form of updated and refined models and save it in an experience base to be reused on future projects.
Quality Improvement Paradigm

Corporate learning

- Set goals
- Choose processes, methods, techniques, and tools
- Package & store experience
- Characterize & understand

Project learning

- Provide process with feedback
- Execute process
- Analyze results

Fraunhofer USA
Center for Experimental Software Engineering
Quality Improvement Paradigm

Project Organization

1. Characterize
2. Set Goals
3. Choose Process

Execution plans

4. Execute Process

Experience Factory

Experience Base

6. Package
   - Generalize
   - Tailor
   - Formalize
   - Disseminate

5. Analyze
   - project analysis, process modification
   - data, lessons learned

4. Execute Process

3. Choose Process

2. Set Goals

1. Characterize

Execution plans

environment characteristics
- tailorable knowledge, consulting
- products, lessons learned, models

Project Support

5. Analyze

4. Execute Process

3. Choose Process

2. Set Goals

1. Characterize

Execution plans

Experience Factory

Experience Base

6. Package
   - Generalize
   - Tailor
   - Formalize
   - Disseminate

5. Analyze
   - project analysis, process modification
   - data, lessons learned

4. Execute Process

3. Choose Process

2. Set Goals

1. Characterize

Experience Factory

Experience Base

6. Package
   - Generalize
   - Tailor
   - Formalize
   - Disseminate

5. Analyze
   - project analysis, process modification
   - data, lessons learned

4. Execute Process

3. Choose Process

2. Set Goals

1. Characterize

Execution plans

Project Organization

1. Characterize
2. Set Goals
3. Choose Process

Execution plans

4. Execute Process

Fraunhofer USA
Center for Experimental Software Engineering
Phase III ~1990s
Tying studies together

Kinds of Questions:

Can we *reduce the risk* in trying out a new approach?

Can we apply a mix of controlled experiments, case studies, quasi-experiments, qualitative analysis, and simple observations tied together to support hypotheses, reduce the threats to validity, and better understand the effect of context?

Are some techniques, methods more effective for different types of defects?
Phase III
Tying studies together

• Situation:
  – Context variables were beginning to be recognized as critical
  – …

• Lessons Learned:
  – Can evolve a process by learning from multiple approaches
  – Different techniques, methods may be more effective for different types of defects
  – Can reduce risk by running smaller experiments off-line
  – Can build confidence in a theory based upon multiple treatments
  – Techniques can be developed based upon specific goals
## Combining Evaluation Approaches

### Running Multiple Studies

#### Experiment Classes

<table>
<thead>
<tr>
<th># of Teams Per Project</th>
<th>#Projects</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>One</td>
<td>Single Project</td>
<td>Multi-Project Variation</td>
</tr>
<tr>
<td>More than one</td>
<td>Replicated Project</td>
<td>Blocked Subject-Project</td>
<td></td>
</tr>
</tbody>
</table>
# Experimental Learning Mechanisms

## Series of Studies

<table>
<thead>
<tr>
<th># of Teams per Project</th>
<th>One</th>
<th>More than one</th>
<th>Reading vs. Testing</th>
</tr>
</thead>
</table>

- Code reading, functional and structural testing
- Unit test size programs seeded with faults
- Blocked subject-project: Fractional factorial design
- Three replications: 42 UM (2), 32 NASA/CSC
Experimental Learning Mechanisms

Series of Studies

Scaled up to teams and larger projects
Compared 15 teams using Cleanroom and not using it
When reading is motivated it is very effective

<table>
<thead>
<tr>
<th># of Teams per Project</th>
<th>One</th>
<th>More than one</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>2. Cleanroom at Maryland</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>1. Reading vs. Testing</strong></td>
</tr>
</tbody>
</table>
Training and tailoring of Cleanroom for the SEL
Integrated into the existing process
Very effective on live project (40K SLOC) at NASA

<table>
<thead>
<tr>
<th># of Teams per Project</th>
<th>One</th>
<th>More than one</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Reading vs. Testing</td>
<td>2. Cleanroom at Maryland</td>
<td>3. Cleanroom (SEL Project 1)</td>
</tr>
</tbody>
</table>
## Experimental Learning Mechanisms

Effective over a series of projects
Some modification for contracted out projects
Recognized need to reading at higher level, e.g. Requirements reading

<table>
<thead>
<tr>
<th># of Teams per Project</th>
<th>One</th>
<th>More than one</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>3. Cleanroom (SEL Project 1)</td>
<td>4. Cleanroom (SEL Projects, 2,3,4,...)</td>
</tr>
<tr>
<td>More than one</td>
<td>2. Cleanroom at Maryland</td>
<td>1. Reading vs. Testing</td>
</tr>
</tbody>
</table>

Fraunhofer USA Center for Experimental Software Engineering
## Experimental Learning Mechanisms

### Series of Studies

Developed perspective based reading techniques  
Experimented with requirements reading  
Effective on controlled experiments with NASA developers

| # of Teams per Project | 1. Reading vs. Testing at Maryland  
2. Cleanroom at Maryland  
3. Cleanroom (SEL Project 1)  
4. Cleanroom Projects, 2,3,4,... | 5. Scenario reading vs. ... |
Phase IV ~2000
Expanding across domains, environments, technologies

Kinds of Questions:

Can we build a body of knowledge about a technology supported by empirical evidence?

Can we build a body of knowledge about a domain supported by empirical evidence?

Can we build a decision framework to evaluate and choose among software development technologies?

Can we create an empirical research engine to provide the empirical evidence of what works and when?
Phase IV
Expanding out across domains, environments, technologies

**Situation:**
Examples of studies building knowledge about a domain
Conferences and Journals opened up to empirical research
The community of researchers continues to expand
There are various kinds of replications being published
There is a rich palate of experimental methods in use
**Context variables** are being studied and characterized and we are building knowledge across different contexts

**Lessons Learned:**
More research to do before we can solve the problems of
  - Building a software engineering research engine
  - Building the decision support system with partial knowledge
  - Integrating the process
CeBASE Project Goal: Enable a decision framework and experience base that forms a basis and infrastructure needed to evaluate and choose among software development technologies.

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

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

Observation and Evaluation Studies of Development Technologies and Techniques

Empirical Data

Predictive Models
(Quantitative Guidance)

General Heuristics
(Qualitative Guidance)

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

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

Technology maturity

Practical applications
(Government, industry, academia)

Applied Research

Basic Research

Primary activities

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

Experimentation and analysis with the concepts in selected areas.

Building a SE Empirical Research Engine and Experience base structure

Evolving results

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

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

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

Fraunhofer USA
Center for Experimental Software Engineering
CeBASE Basic Research Activities

Define and improve methods to

- Formulate evolving hypotheses regarding software development decisions
- Collect empirical data and experiences
- Record influencing variables (context)
- 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.W, Fraunhofer-MD, …
What are the top level research problems?

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

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

**Research Problem 3**
What set of technologies should be applied to achieve the desired quality? (Decision Support)
UMD is a model builder

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

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

**Issue**
- Failure
  - Type
    - Accuracy
    - Response Time
    - etc.
  - Availability impact
    - Stopping
    - Non-Stopping
  - Severity
    - High
    - Low

- Hazard
  - Severity
    - People affected
    - Property only
    - etc.

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

**Reaction**
- Impact mitigation
  - Warnings
  - Alternative services
  - Mitigation services
- Recovery
  - Recovery time / actions
- Occurrence reduction
  - Guard services

**Concern**
- Type
- Adverse Condition
- Attack
- etc.
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
Applied Research
DARPA High Productivity Computing Systems

**Problem:** How do you build sufficient knowledge about the high end computing (HEC) so you can improve the time and cost of developing these 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

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

**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
Studies Conducted

- UCSB: 3 studies
- USC: 4 studies
- UCSD: 1 study
- CalTech: ASCI Alliance
- U Utah: ASCI Alliance
- UIUC: ASCI Alliance
- U Chicago: ASCI Alliance
- MIT: 3 studies
- UMD: 6 studies
- Mississippi State: 2 studies
- Iowa State: 1 study
- Stanford U: ASCI Alliance
- UIUC: ASCI Alliance
- U Chicago: ASCI Alliance
- U Utah: ASCI Alliance
- UCSD: 1 study
- CalTech: ASCI Alliance
- UMD: 6 studies
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.
Phase V now and the future

Situation:
The ESE community has evolved
Many collaborate across international boundaries
Shared authorship across institutions on many papers
22nd ISERN workshop this year
19th year of the Journal of Empirical Software Engineering with an ISI rating on a par with IEEE-TSE and ACM-TOSEM
Most journals now welcome empirical work, in fact many expect it
ACM/IEEE Empirical Software Engineering and Measurement Symposium in its 12th year
There are textbooks in the field
Phase IV now and the future

But:

Not enough use of the scientific method, i.e., learning from applying, outside the ESE community

Identifying and accounting for context variables still a problem

Long term funding is necessary but very difficult to acquire

Collaborations with ‘laboratories’ is not easy to do
What is needed

Software engineering requires an empirical research engine that identifies the benefits, limits, and bounds of technologies.

Have come a long way in evolving the discipline of empirical software engineering, but we have a long way to go.

Software engineering does not have an empirical culture. CS departments were mostly spawned by mathematics departments and mathematics is not an empirical science. So we did not inherit an empirical mind set.

The building of our research engine is at its infancy. It needs to be better understood not just by empiricists but by software engineers in general.
What is needed

Software engineering needs to codify the relationships between processes and products

We need to build a tapestry of models and guidelines that represent our knowledge about the benefits and the bounds and limits of techniques, methods, and life cycle models as well as models that take into account product characteristics of all kinds.

The real question is: if I want a product to have certain characteristics, e.g., reliability, correctness, safety, security, etc. what are the appropriate techniques methods, life-cycle models to achieve those characteristics?
What is needed

Empirical software engineering needs to balance the symbiotic relationship between theory and practice

If we are to make more progress in the discipline of software engineering in both practice and research, the relationship between practice and research has to be nourished so both groups can gain and the discipline can evolve.

We need many applications of a process, taking place in different environments, each application providing a better understanding of the concepts and their interaction in context

Over time, all context variables need to be considered
What is needed

Empirical studies in software engineering need multi-disciplinary teams

Research teams need various forms of expertise, e.g., domain knowledge, software engineering knowledge, a variety of experimentation capabilities

The team not only provides different levels of expertise but provides checks and balances on the study itself
What is needed

Replication in software engineering studies is critical but it should be expanding knowledge rather than confirming it.

Replication in ESE should play the role of expanding our understanding of the context variables in which existing results may or may not be true.

It requires close interaction between the original study team and the replication team, because we cannot always communicate the original context variables.

It is hard enough to capture tacit knowledge in replicating experiments, even when the teams are collaborating.
What is needed

**Empirical software engineering needs to build collaborative, communicating communities**

Building the tapestry of software engineering knowledge is too big a task for any one group to perform.

Empirical software engineering requires groups who share results in effective ways, e.g., via a repository of evolving models and lessons learned that can be used, added to, and evolved by other researchers.

For each group, the focus can be bounded, limiting the context, the domain, the collection of techniques, methods, and life cycle models studied.
What is needed

Empirical SE needs to build and evolve a shared experience base representing what we know about the discipline at any point in time and decision support system that provides support for the practice of software engineering.

The internal information is the same but the interfaces are different geared to different populations.

A shared repository of evolving models and lessons learned that can be effectively added to and used by researchers.

*Long term support* across organizations /countries of collaborators.

A reward system for researchers to contribute and share.
Summary

Building a discipline of software engineering requires

• many collaborations with long term goals and longitudinal studies
• the development of a framework for communicating, coordinating, and integrating experiential models with long term support that will exist and be available to capture all forms of evidence
• Long term support

We need to learn from applying the discipline to discover whether relationships hold, how they vary, and what the limits of various technologies are, so we can know how to configure processes to develop software better

Software engineering is big science
Software Engineering is “big science”; not small independent technology developments.