Experimentation in Software Engineering

Victor R. Basili

Experimental Software Engineering Group

Institute for Advanced Computer Studies

Department of Computer Science

University of Maryland

and

Fraunhofer Center for Experimental Software Engineering - Maryland

Motivation

- Software development teams need to understand the right models and techniques to support their projects. For example:
  - When are peer reviews more effective than functional testing?
  - When should you use a procedural approach to code reviewing?
  - How should you tailor a lifecycle model for your environment?
- Too often, such decisions are based on anecdote, hearsay, or hype
- How do other disciplines build knowledge about
  - the elements of their discipline, e.g., their products and processes
  - the relationships between those elements

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

Software Engineering
The Nature of the Discipline

Like other disciplines, software engineering requires the cycle of model building, experimentation, and learning

Software engineering is a **laboratory science**

The **researcher's role** is to understand the nature of the processes, products and the relationship between the two 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**

What are the problems of interest in software engineering?

Practitioners want

- the ability to control and manipulate project solutions
  - based upon the environment and goals set for the project
- knowledge based upon empirical and experimental evidence
  - of what works and does not work and under what conditions

Researchers want to understand

- the basic elements of the discipline, e.g., products, processes, and their characteristics (build realistic models)
- the variables associated with the models of these elements
- the relationships among these models

Researchers need laboratories for experimentation

This will require a research plan that will take place over many years

- coordinating experiments
- evolving with new knowledge

Software Engineering
Early Observation

Belady & Lehman ('72,'76)

- **observed** the behavior of OS 360 with respect to releases
- posed theories based on observation concerning entropy

The idea that you might redesign a system rather than continue to change was a revelation

it

But, Basili & Turner ('75)

- **observed** that a compiler system
- being developed using an incremental development approach
- gained structure over time, rather than lost it

How can these **seemingly** opposing statements be true?

What were the variables that caused the effects to be different?

Size, methods, nature of the changes, context?

Software Engineering
Early Observation

Walston and Felix ('79) identified **29** variables that had an effect on software productivity in the IBM environment

Boehm ('81) observed that **15** variables seemed sufficient to explain/predict the cost of a project across several environments

Bailey and Basili ('81) identified **2** composite variables that when combined with size were a good predictor of effort in the SEL environment

There are numerous cost models with different variables

Why were the variables different?

What does the data tell us about the relationship of variables?

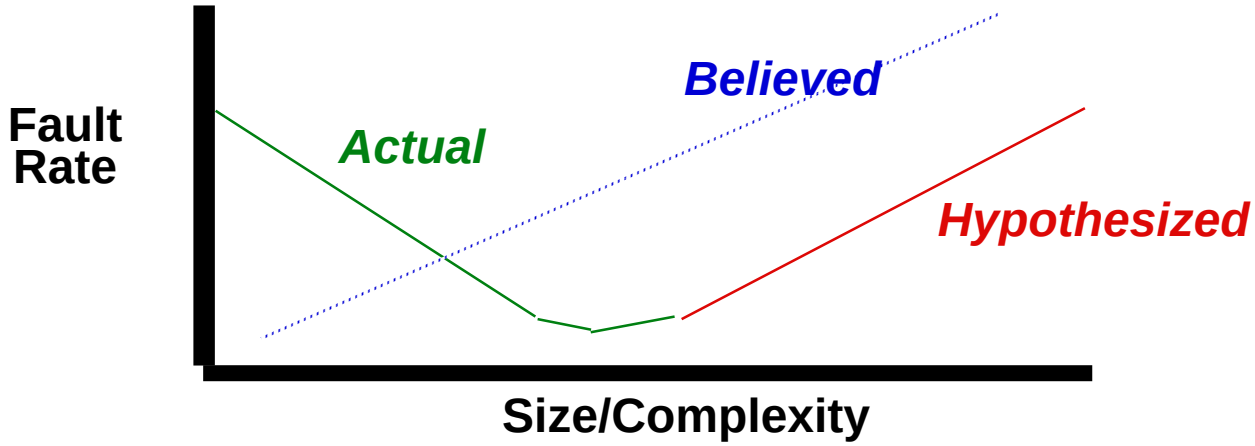
Which variable are relevant for a particular context?

What determines their relevance?

What are the ranges of the values variables and their effects?

Software Engineering
Early Observation

Basili & Perricone (‘84) **observed** that the defect rate of modules shrunk as module size and complexity grew in the SEL environment



Seemed counter to folklore that smaller modules were better, **but**

- interface faults dominate
- developer tend to shrink size when they lose control

This result has been observed by numerous other organizations

But **defect rate is only one dependent variable**

What is the effect on other variables? What size minimizes the defect rate?

Available Research Paradigms

Experimental paradigm:

- observing the world (or existing solutions)
- proposing a model or a theory of behavior (or better solutions)
- measuring and analyzing
- validating hypotheses of the model or theory (or invalidate
- repeating the procedure evolving our knowledge base

The experimental paradigms involve

- experimental design
- observation
- quantitative or qualitative analysis
- data collection and validation on the process or product being studied

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

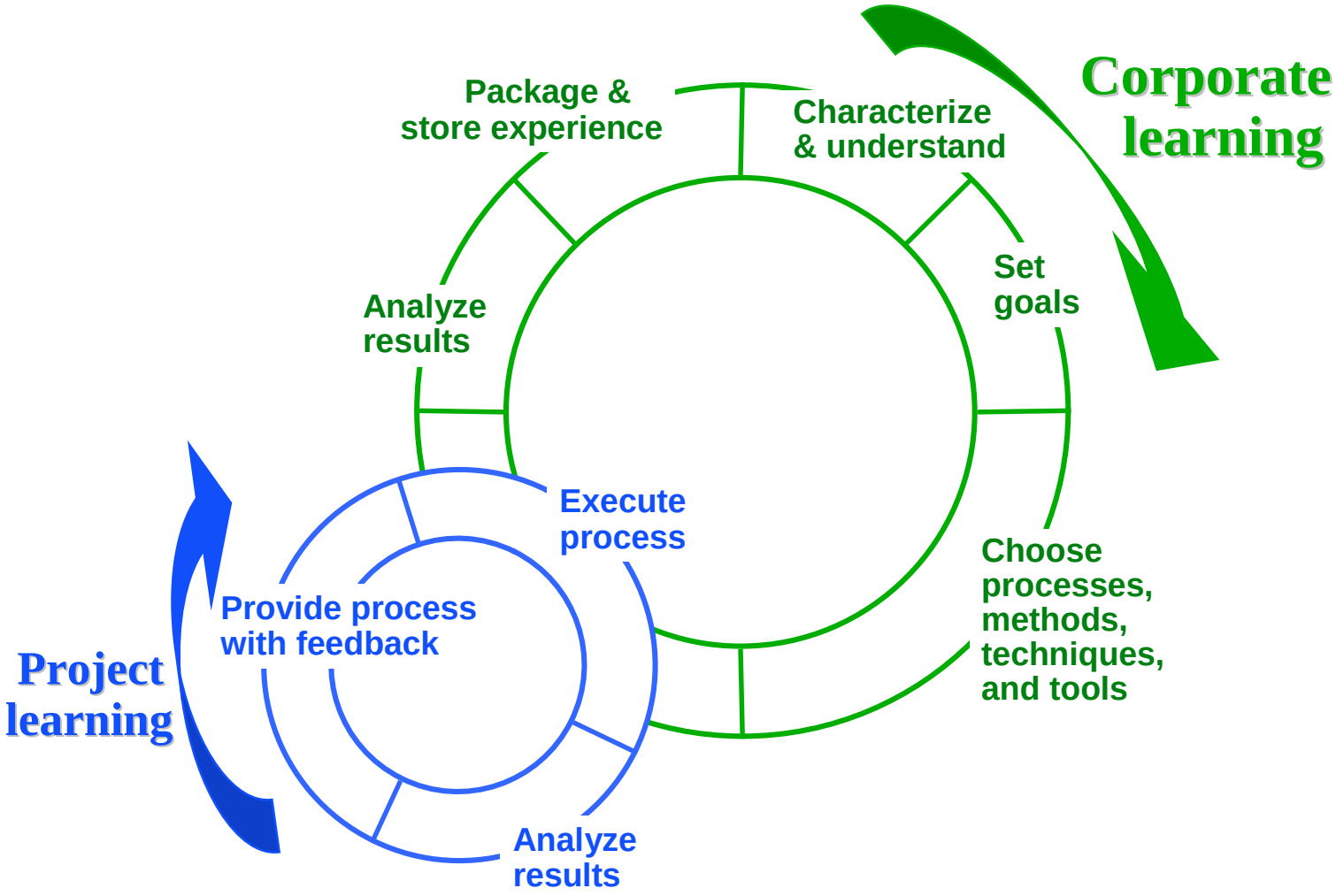
Choose the appropriate project **processes**, supporting methods and tools

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

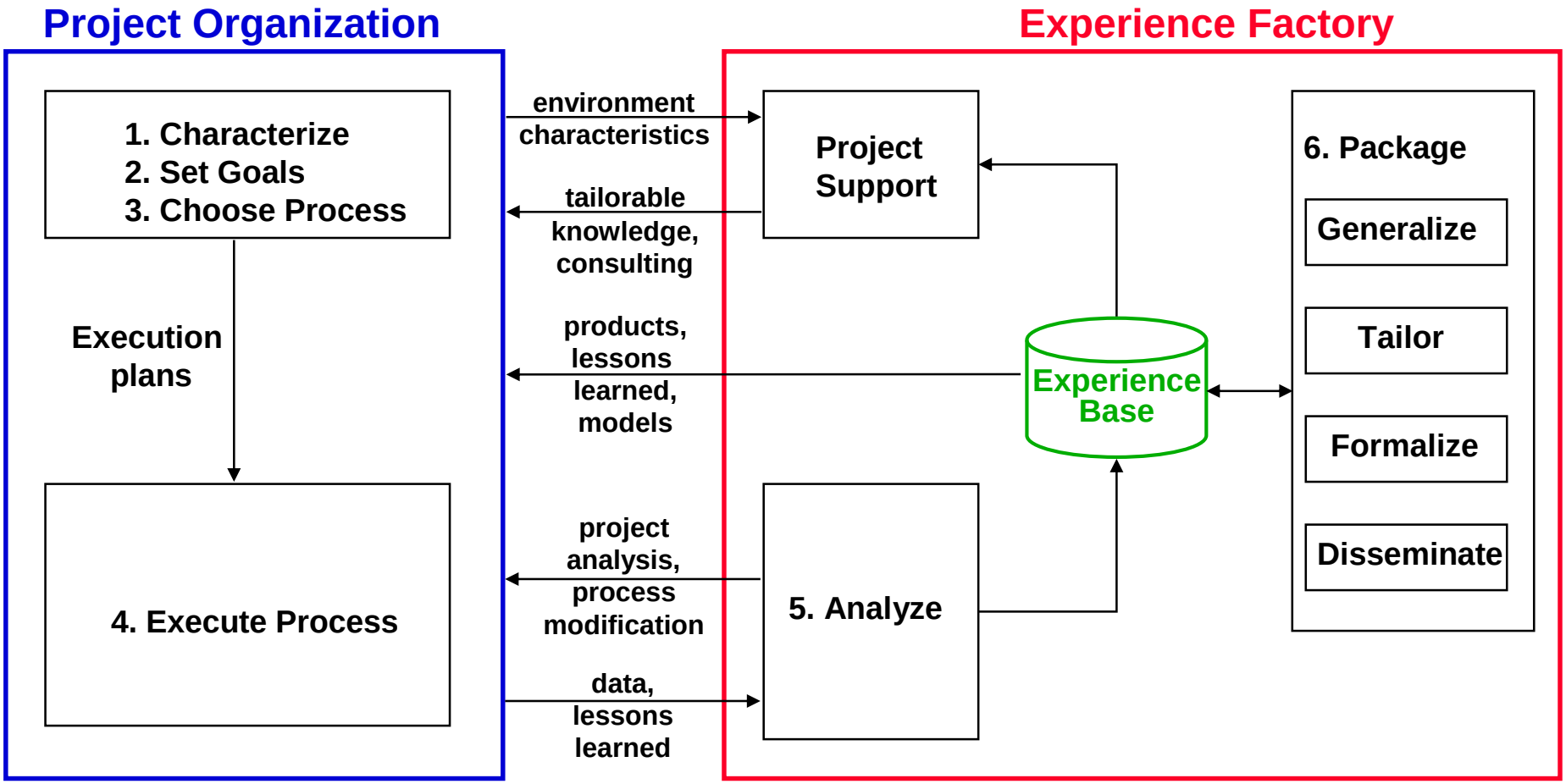
Analyze the **data** to evaluate current practices, determine problems, record findings, recommend improvements for future project

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



The Experience Factory Organization



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**

Software Models and Measures

Perspectives

Characterize

Describe and differentiate software processes and products

Build descriptive models and baselines

Understand

Explain associations/dependencies between processes and products

Discover causal relationships

Analyze models

Evaluate

Assess the achievement of quality goals

Assess the impact of technology on products

Compare models

Predict

Estimate expected product quality and process resource consumption

Build predictive models

Motivate

Describe what we need to do to control and manage software

Build prescriptive models

The Status of Model Building

Resource Models and Baselines,

e.g., local cost models, resource allocation models

Change and Defect Baselines and Models,

e.g., defect prediction models, types of defects expected for application

Product Models and Baselines,

e.g., actual vs. expected product size and library access over time

Process Definitions and Models,

e.g., process models for Cleanroom, Ada

Method and Technique Evaluations,

e.g., best method for finding interface faults

Products, e.g., Ada generics for simulation of satellite orbits

Quality Models,

e.g., reliability models, defect slippage models, ease of change models

Lessons Learned, e.g., risks associated with an Ada development

The Status of the Experimental Discipline

What kinds of studies have been performed?

Experiments can be
- controlled experiments
- quasi-experiments or pre-experimental designs

Controlled experiments, typically:
- small object of study
- in vitro
- a mix of both novices (mostly) and expert treatments

Sometimes, novice subjects used to “debug” the experimental design

Quasi-experiments or Pre-experimental design, typically:
- large projects
- in vivo
- with experts

These experiments tend to involve a qualitative analysis component, including at least some form of interviewing

The Maturing of the Experimental Discipline

What kinds of studies have been performed?

Experiment Classes

		#Projects	
		One	More than one
# of Teams per Project	One	Single Project	Multi-Project Variation
	More than one	Replicated Project	Blocked Subject-Project

The Maturing of the Experimental Discipline

How is experimentation maturing?

Signs of maturity in a field:

level of sophistication of the goals of an experiment
understanding interesting things about the discipline
a **pattern of knowledge** built from a **series of experiments**

Researchers appear to be

- asking more **sophisticated questions**
- **studying relationships** between processes/product characteristics
- doing more studies **in the field** than in the laboratory
- **combining** various **experimental classes** to build knowledge

An Example:
The Evolution of Reading Techniques

We discuss a series of experiments
- begun at the University of Maryland and at NASA used to learn about, evaluate, and evolve reading techniques
- replicated and evolved at various sites around the world

This example
- shows **multiple experimental designs**
- provides a combination of **evaluation approaches**
- offers insight into the **effects of different variables** on reading

The experiments start with
the early reading vs. testing experiments
to various Cleanroom experiments
to the scenario based reading techniques currently under study

An Example:
The Evolution of Reading Techniques

Reading is a **key technical activity** for analyzing and constructing software artifacts

Reading is **a model for writing**

Reading is **critical for reviews, maintenance, reuse, ...**

What is a reading technique?

a concrete set of instructions given to the reader saying how to read and what to look for in a software product

More Specifically, software reading is

the individual analysis of a software artifact

e.g., requirements, design, code, test plans

to achieve the understanding needed for a particular task

e.g., defect detection, reuse, maintenance

An Example:
The Evolution of Reading Techniques

Series of Studies

		# Projects	
		One	More than one
# of Teams	One	3. Cleanroom (SEL Project 1)	4. Cleanroom (SEL Projects, 2,3,4,...)
per Project	More than one	2. Cleanroom at Maryland	1. Reading vs. Testing 5. Scenario reading vs. ...

EXPERIMENT
Blocked Subject Project Study

Analysis Technique Comparison

Technique Definition:

Code Reading vs **Functional Testing** vs **Structural Testing**

Compare with respect to:

fault detection effectiveness and cost

classes of faults detected

Experimental design:

Fractional factorial design

Environment:

University of Maryland (43) and then NASA/CSC (32)

Module size programs (145 - 365 LOC), seeded with faults

Cause-effect, in vitro, novices and experts

Blocked Subject Project Study
Testing Strategies Comparison

Fractional Factorial Design

		Code Reading			Functional Testing			Structural Testing		
		P1	P2	P3	P1	P2	P3	P1	P2	P3
Advanced Subjects	S1			X		X		X		
	S2		X		X					X
	:									
	S8	X					X		X	
Intermediate Subjects	S9			X		X		X		
	S10		X		X					X
	:									
	S19	X					X		X	
Junior Subjects	S20			X		X		X		
	S21		X		X					X
	:									
	S32	X					X		X	

Blocking by experience level and program tested

NASA/CSC

Blocked Subject Project Study
Analysis Technique Comparison

Some Results (NASA/CSC)

Code reading more
- **effective** than functional testing
- **efficient** than functional or structural testing

Different techniques more effective for different defect classes
- code reading more effective for interface defects
- functional testing more effective for control flow defects

Code readers assessed the true quality of product better than testers

<u>After completion of study:</u>
- Over 90% of the participants thought functional testing worked best

Some Lessons Learned

Reading is effective/efficient; the particular technique appears important

The choice of techniques should be tailored to the defect classification

Developers don't believe reading is better

Blocked Subject Project Study
Analysis Technique Comparison

Based upon this study

reading was implemented as part of the SEL development process

But - reading appeared to have very little effect

Possible Explanations (NASA/CSC)

Hypothesis 1: People did not read as well as they should have as they believed that testing would make up for their mistakes

Experiment: If you read and cannot test you do a more effective job of reading than if you read and know you can test.

Hypothesis 2: there is a confusion between the reading technique and the reading method

NEXT: Is there an approach with reading motivation and technique?

⇨ Try Cleanroom in a controlled experiment at the University of Maryland

EXPERIMENT
Replicated Project Study

Cleanroom Study

Approaches:

Cleanroom process vs. **non-Cleanroom process**

Compare with respect to:

effects on the process product and developers

Experimental design:

15 three-person teams (10 teams used Cleanroom)

Environment:

University of Maryland

Electronic message system, ~ 1500 LOC

novice, in vitro, cause-effect

Replicated Project Study
Cleanroom Evaluation

Some Results

Cleanroom developers

- more effectively applied off-line review techniques
- spent less time on-line and used fewer computer resources
- made their scheduled deliveries

Cleanroom product

- less complex
- more completely met requirements

Some Lessons Learned

Cleanroom developers were motivated to read better

Cleanroom/Reading by step-wise abstraction was effective and efficient

NEXT: Does Cleanroom scales up? Will it work on a real project?
Can it work with changing requirements?
⇨ Try Cleanroom in the SEL

EXPERIMENT
Single Project Study

First Cleanroom in the SEL

Approaches:

Cleanroom process vs. **Standard SEL Approach**

Compare with respect to:

effects on the effort distribution, cost, and reliability

Experimental design:

Apply to a live flight dynamics domain project in the SEL

Environment:

NASA/ SEL
40 KLOC Ground Support System
in vivo, experts, descriptive

Single Project Study
First Cleanroom in the SEL

Some Results

Cleanroom was

- effective for 40KLOC
  - failure rate reduced by 25%
  - productivity increased by 30%
  - less computer use by a factor of 5
- usable with changing requirements
  - rework effort reduced
    - 5% as opposed to 42% took > 1 hour to change

Some Lessons Learned

Cleanroom/Reading by step-wise abstraction was effective and efficient
Reading appears to reduce the cost of change

Better training needed for reading methods and techniques

NEXT: Will it work again? Can we scale up more? Can we contract it out?

⇨ Try on larger projects, contracted projects

EXPERIMENT
Multi-Project Analysis Study

Cleanroom in the SEL

Approaches:

Revised Cleanroom process vs. **Standard SEL Approach**

Compare with respect to:

effects on the effort distribution, cost, and reliability

Experimental design:

Apply to three more flight dynamics domain projects in the SEL

Environment:

NASA/ SEL

Projects: 22 KLOC (in-house)
160 KLOC (contractor)
140 KLOC (contractor)

in vivo, experts, descriptive

Multi-Project Analysis Study
Cleanroom in the SEL

Major Results

Cleanroom was

- effective and efficient for up to ~ 150KLOC
- usable with changing requirements
- took second try to get really effective on contractor, large project

Some Lessons Learned

Cleanroom Reading by step-wise abstraction

- effective and efficient in the SEL
- takes more experience and support on larger, contractor projects
- appears to reduce the cost of change

Unit test benefits need further study

Better training needed for reading techniques

Better techniques for other documents, e.g., requirements, design, test plan

NEXT: Can we improve the reading techniques for requirements and design documents?

⇨ Develop reading techniques and study effects in controlled experiments

Scenario-Based Reading Definition

An approach to generating a family of reading techniques, called **operational scenarios**, has been defined to be

- document and notation specific
- tailorable to the project and environment
- procedurally defined
- goal driven
- focused to provide a particular coverage of the document
- empirically verified to be effective for its use
- usable in existing methods, such as inspections

- These goals defines a set of guidelines/characteristics for a process definition for reading techniques that can be studied experimentally

Scenario-Based Reading Definition

So far, we have developed five families of reading techniques parameterized for use in different contexts and evaluated experimentally in those contexts

They include:

perspective based reading:

for detecting defects in **requirements documents in English**

defect based reading:

for detecting defects in **requirements documents in SCR**

scope based reading:

for constructing designs from **OO frameworks**

use based reading:

for detecting anomalies in **user interface web screens**

horizontal/vertical reading:

for detecting defects in **object oriented design in UML**

EXPERIMENTING
Blocked Subject-Project Study

Scenario-Based Reading

Approaches:
defect-based reading vs **ad-hoc reading** vs **check-list reading**

Compare with respect to:
fault detection effectiveness in the context of an inspection team

Experimental design:
Partial factorial design
Replicated twice
Subjects: 48 subjects in total

Environment:
University of Maryland
Two Requirements documents in SCR notation
Documents seeded with known defects
novice, in vitro, cause-effect

EXPERIMENTING
Blocked Subject Project Study

Scenario-Based Reading

Approaches:

perspective-based reading vs **NASA's reading technique**

Compare with respect to:

fault detection effectiveness in the context of an inspection team

Experimental design:

Partial factorial design
Replicated twice
Subjects: 25 subjects in total

Environment:

NASA/CSC SEL Environment
Requirements documents:
Two NASA Functional Specifications
Two Structured Text Documents
Documents seeded with known defects
expert, in vitro, cause-effect

Blocked Subject Project Study
Scenario-Based Reading

Some Results

Scenario-Based Reading performed better than
Ad Hoc, Checklist, NASA Approach reading
especially when they were less familiar with the domain

Scenarios helped reviewers focus on specific fault classes
but were no less effective at detecting other faults

The relative benefit of Scenario-Based Reading is higher for teams

Some Lessons Learned

Need better tailoring of Scenario-Based Reading to the NASA environment in terms of document contents, notation and perspectives

Need better training to stop experts from using their familiar technique

Next: Replicate these experiments in many different environments
- varying the context

The Maturing of the Experimental Discipline

How is experimentation maturing?

Several of these **experiments have been replicated**

- under the same and differing contexts
- at a variety of organizations in different countries, e.g.,
  - University of Kaiserslautern, Germany
  - University of Bari, Italy
  - University of New South Wales, Australia
  - Bell Laboratories, USA
  - University of Trondheim, Norway
  - Bosch, Germany
  - Univerities of Sao Paolo and Rio deJaniero, Brazil

to better understand the reading variable

ISERN

organized explicitly to share knowledge and experiments
has membership in the U.S., Europe, Asia, and Australia
represents both industry and academia
supports the publication of artifacts and laboratory manuals

Its goal is to evolve software engineering knowledge over time, based upon experimentation and learning

Evolving Knowledge
Model Building, Experimenting, and Learning

Abstracting across Reading Experiments

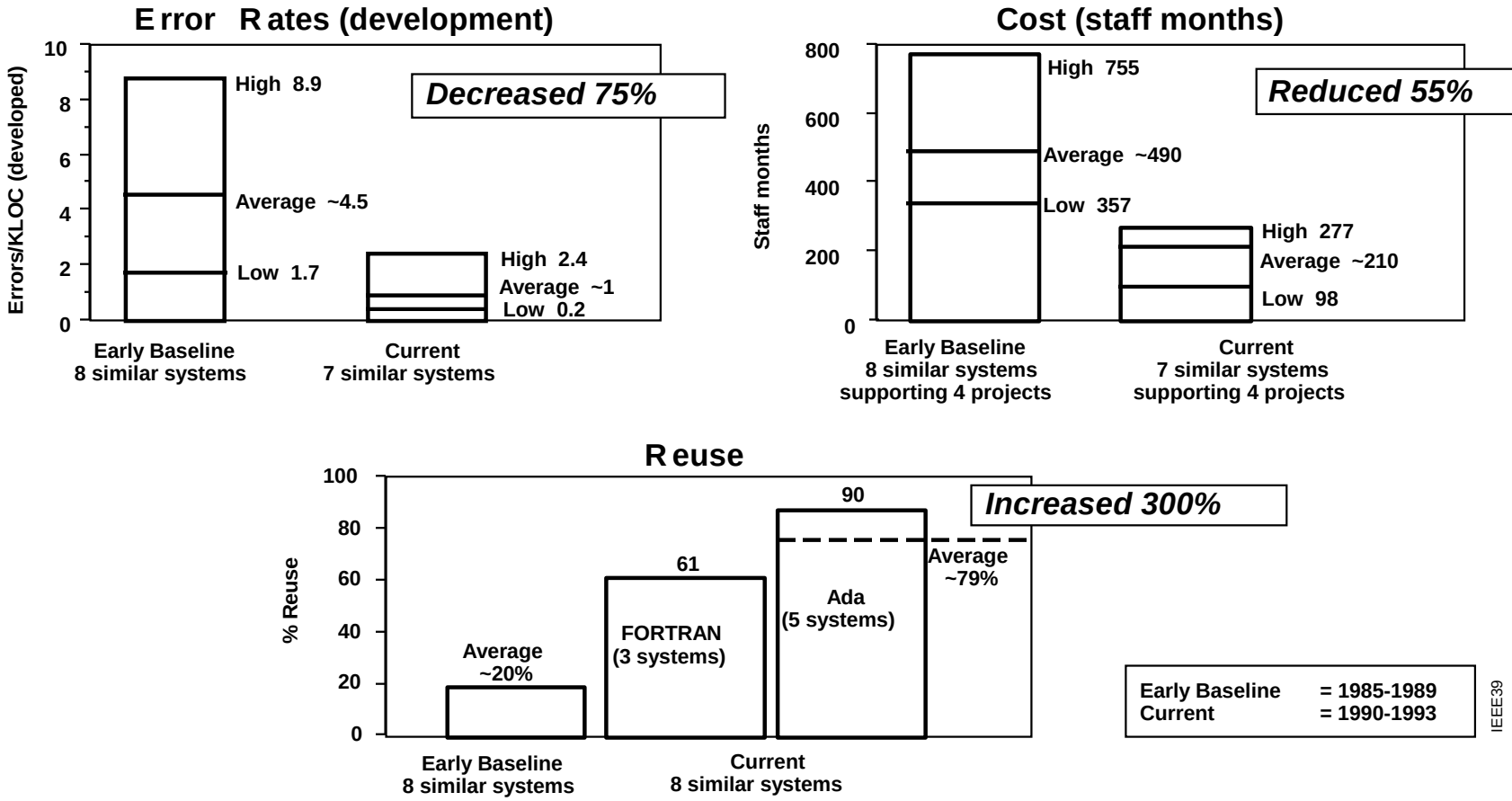
We have generated useful **empirical results for technique definition guidance**

- For a reviewer with an average experience level, a **procedural approach** to defect detection is more effective than a less procedural one.

- Procedural inspections, based upon **specific goals**, will find defects related to those goals, so inspections can be customized.

- A set of readers of a software artifact are more effective in uncovering defects when each uses a **different and specific focus.**

Using Experimentation to Improve Software at NASA/GSFC



The Software Engineering Laboratory was awarded the first
IEEE Computer Society Award for Software Process Achievement in 1994
for demonstrable, sustained, measured, significant process improvement

Using Experimentation to Improve Software at NASA/GSFC

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 - Maryland created 1998

CeBaSE

Center for Empirically-Based Software Engineering created 2000

Evolving Knowledge
Model Building, Experimenting, and Learning

CeBASE Project: Vision and Approach

The goal of the Center for empirically-Based Software Engineering (CeBASE) is to accumulate **empirical models** to provide validated guidelines for selecting techniques and models, recommend areas for research, and support education

A first step is to build an empirical **experience base**

continuously evolving with empirical evidence

to help us identify what affects cost, reliability, schedule,...

To achieve this we are

- Integrating existing data and models
- Initially focusing on new results in two high-leverage areas
  - Defect Reduction, e. g. reading techniques (see top ten issues)
  - COTS Based Development (see top ten issues)

Examples of Using Empirical Results for development, research, education

Technique Tailoring

Is tailoring the reading process associated with an inspection worth the effort?

- Procedural inspections, based upon specific goals, will find defects related to those goals, so inspections can be customized. (UMD)

Implications for empirically based software development process:

- The better you can articulate your goals, the more effectively you can choose and tailor process.

Implications for software engineering research:

- It is important to empirically study the effects of processes on product

Implications for software engineering education:

- Don't teach that there is a one size fits all process; teach how to tailor processes

Examples of Using Empirical Results for development, research, education

Technique Selection Guidance

When should you use a procedural approach to code reviewing?

- For a reviewer with an average experience level, a **procedural approach** to defect detection is more effective than a less procedural one (UMD, USC)

Implications for empirically based software development process:

- Experts might be more effective working on their own but most people should apply a procedural approach. Novices need training.

Implications for software engineering research:

- How can we improve document reading procedures based upon how experts analyze documents?

Implications for software engineering education:

- Effective procedures that can be taught for reviewing documents

Examples of Using Empirical Results for development, research, education

Technique Selection Guidance

When are peer reviews more effective than functional testing?

- **Peer reviews** are more effective than functional testing for faults of **omission** and **incorrect specification** (UMD, USC)

Implications for empirically based software development process:

- If, for a given project set, there is an expectation of a larger number of faults of omission or incorrect facts than use peer reviews.

Implications for software engineering research:

- How can peer reviews be improved with better reading techniques for faults of omission and incorrect fact?

Implications for software engineering education:

- Teach how to experiment with and choose the appropriate analytic techniques

Examples of Useful Empirical Results

Lifecycle Selection Guidance

Lifecycle Selection Guidance

- The **sequential waterfall model** is suitable if and only if
  - The **requirements** are **knowable** in advance,
  - The **requirements** have no **unresolved**, high-risk implications,
  - The **requirements satisfy** all the key stakeholders' **expectations**,
  - A **viable architecture** for implementing the requirements is **known**,
  - The **requirements** will be **stable** during development,
  - There is **enough calendar time** to proceed sequentially. (USC)
- The **evolutionary development model** is suitable if and only if
  - The **initial release** is **good** enough to keep the key stakeholders involved,
  - The **architecture** is **scalable** to accommodate needed system growth,
  - The operational user organizations can adapt to the **pace of evolution**,
  - The **evolution dimensions** are **compatible** with legacy system replacement,
  - **appropriate** management, financial, and incentive **structures are in place**. (USC)

Conclusion

The software engineering discipline needs to

build **software core competencies** as part of overall **business strategy**
create **organizations for continuous learning** to improve software competence
generate a tangible **corporate asset**: an **experience base of competencies**
build an **empirically-based, tailorable software development process**

Experimentation can provide us with

- better **scientific and engineering basis** for the software
- **better models** of software processes, products, and environmental factors
- better **understanding of the interactions** of these models

Practitioners are provided with

- the ability to control and manipulate project solutions
- knowledge of what works and does not work and under what conditions

Researchers are provided laboratories for experimentation