Evaluation of SE research

- What convinces you?
- Why?

Possible answers include

- Intuition
- Quantitative assessments
- Qualitative assessments
- Case studies
- ... other possible answers?

Brooks on evaluation

- The first user gives you infinite utility — that is, you learn more from the first person who tries an approach than from every person thereafter.
- In HCI, Brooks compared
  - "narrow truths proved convincingly by statistically sound experiments, and
  - broad 'truths', generally applicable, but supported only by possibly unrepresentative observations."

More on Brooks by Mary Shaw

Brooks proposes to relieve the tension through a certainty-shell structure—
to recognize three nested classes of results,
- Findings: well-established scientific truths, judged by truthfulness and
  rigor;
- Observations: reports on actual phenomena, judged by interestingness;
- Rules of thumb: generalizations, signed by their author but perhaps
  incompletely supported by data, judged by usefulness.

What Makes Good Research in Software Engineering? International Journal of
Software Tools for Technology Transfer, 2002

Shaw: research questions in SE

<table>
<thead>
<tr>
<th>Type of question</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method of analysis</td>
<td>How can we discover (or automate doing) X?</td>
</tr>
<tr>
<td>Method of evaluation</td>
<td>What is a better way to discover X?</td>
</tr>
<tr>
<td>Method for analysis</td>
<td>How can I evaluate the quality of correctness of X?</td>
</tr>
<tr>
<td>Method of evaluation</td>
<td>How do I choose between X and Y?</td>
</tr>
<tr>
<td>Design, evaluation, or analysis of a particular instance</td>
<td>What is X, Y, and Z?</td>
</tr>
<tr>
<td>Design, evaluation, or analysis of a particular instance</td>
<td>What is the current state of X, practice of Y?</td>
</tr>
<tr>
<td>Generalization or characterization</td>
<td>What, exactly, do we mean by X?</td>
</tr>
<tr>
<td>Generalization or characterization</td>
<td>What are the important characteristics of X?</td>
</tr>
<tr>
<td>Generalization or characterization</td>
<td>What is a good formal/empirical model for X?</td>
</tr>
<tr>
<td>Generalization or characterization</td>
<td>What are the qualities of X, how are they related?</td>
</tr>
<tr>
<td>Feasibility</td>
<td>Does X even exist, and if so what is it like?</td>
</tr>
<tr>
<td>Feasibility</td>
<td>Is it possible to accomplish X at all?</td>
</tr>
</tbody>
</table>

Shaw: types of SE results

<table>
<thead>
<tr>
<th>Type of result</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure or technique</td>
<td>New or better way to do some task, such as design, implementation, measurement, evaluation, selection from alternatives. Includes operational techniques for implementation, representation, management, and analysis.</td>
</tr>
<tr>
<td>Qualitative or descriptive model</td>
<td>System or system for a problem area, architectural style, framework or design process, or formal basis for analysis.</td>
</tr>
<tr>
<td>Empirical model</td>
<td>Empirical predictive models based on observed data.</td>
</tr>
<tr>
<td>Analytic model</td>
<td>Structured model to support formal analysis or automatic manipulation.</td>
</tr>
<tr>
<td>Notation or tool</td>
<td>Formal language to support technique or model (should have a calculus, semantics, or other basis for computing or inference).</td>
</tr>
<tr>
<td>Specific solution</td>
<td>Solution to applications problems that shows use of software engineering principles—may be design, rather than implementation.</td>
</tr>
<tr>
<td>Framework or judgment</td>
<td>Result of a specific analysis, evaluation, or comparison.</td>
</tr>
</tbody>
</table>

Shaw

- Types of validation

<table>
<thead>
<tr>
<th>Type of validation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Have analyzed my result and found it satisfactory through: small analysis, expert derivation and proof (implied model), data model, controlled use (model).</td>
</tr>
<tr>
<td>Experience</td>
<td>My result has been used as real example by someone other than me, and the evidence of its correctness, usefulness, effectiveness (abstract model), test set, empirical model, data, usually statistics, or practice (reference tool) comparison of this with similar results in techniques, acted on.</td>
</tr>
<tr>
<td>Example</td>
<td>There is an example of how it works on (key example) a key example, perhaps contained by small.</td>
</tr>
<tr>
<td>Life cycle</td>
<td>A system that I have been developing.</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Given the stated reason, can we: (descriptive model) significantly describe the phenomena of interest? (qualitative model) accounts for the phenomena of interest? (empirical model) able to predict because... or gives results that the real data...</td>
</tr>
<tr>
<td>Permanence</td>
<td>Thought hard about this, and believe (technique) if you do the following... system constructed like this model seems reasonable.</td>
</tr>
<tr>
<td>Verification</td>
<td>No serious attempt to evaluate result.</td>
</tr>
</tbody>
</table>
Tichy et al. on quantitative evaluation

- Tichy, Lukowicz, Prechelt & Heinz

Abstract:
A survey of 400 recent research articles suggests that computer scientists publish relatively few papers with experimentally validated results. The survey includes complete volumes of several refereed computer science journals, a conference, and 50 titles drawn at random from all articles published by ACM in 1993. The journals of Optical Engineering (OE) and Neural Computation (NC) were used for comparison. (cont)

Of the papers in the random sample that would require experimental validation, 40% have none at all. In journals related to software engineering, this fraction is 50%. In comparison, the fraction of papers lacking quantitative evaluation in OE and NC is only 15% and 12%, respectively. Conversely, the fraction of papers that devote one fifth or more of their space to experimental validation is almost 70% for OE and NC, while it is a mere 30% for the computer science (CS) random sample and 20% for software engineering. The low ratio of validated results appears to be a serious weakness in computer science research. This weakness should be rectified for the long-term health of the field. The fundamental principle of science, the definition almost, is this: the sole test of the validity of any idea is experiment. —Richard P. Feynman.
Beware of bugs in the above code; I have only proved it correct, not tried it. —Donald E. Knuth

Technology transfer: briefly

- Not a consumer problem
- Not a producer problem
- An ecosystem issue

Evolving the High Performance Computing and Communications Initiative to Support the Nation’s Information Infrastructure (1995)
“Brooks-Sutherland” report
Computer Science and Telecommunications Board (CSTB)
Software engineering economics

- The phrase dates to around 1981, when Barry Boehm published his tome with the same title.
- Boehm identified engineering economics as one “scientific principle” in which software engineering fell short of hardware engineering.
- To the first order, the focus of his book was on how to better estimate effort, cost and schedule for large software projects — COCOMO (COstructive COst MOdel).

COCOMO basics

- Algorithmic software cost estimation modeled with a regression formula that has parameters derived from historical project data and current project characteristics.
- The basic COCOMO equations take the form:
  - Effort Applied = a(KLOC) b (person-months)
  - Development Time = c(Effort Applied) d (months)
  - People required = Effort Applied / Development Time (count)

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>2.4</td>
<td>1.05</td>
<td>2.5</td>
<td>0.38</td>
</tr>
<tr>
<td>Semi-detached</td>
<td>3.0</td>
<td>1.12</td>
<td>2.5</td>
<td>0.35</td>
</tr>
<tr>
<td>Embedded</td>
<td>3.6</td>
<td>1.20</td>
<td>2.5</td>
<td>0.32</td>
</tr>
</tbody>
</table>

Regression parameters
Basic COCOMO

- Based on waterfall-based 63 projects at TRW Aerospace.
- Projects from 2KLOC to 100KLOC, languages from assembler to PL/I.
- The Basic Model designed for rough order-of-magnitude estimates, focused on small to medium-sized projects.
  - Three sets of parameters: organic, semi-detached and embedded.

Intermediate COCOMO

- Uses more parameters (cost drivers) that account for additional differences estimates.
- Product attributes: required software reliability, complexity of the product, ...
- Hardware attributes: run-time performance constraints, memory constraints, ...
- Personnel attributes: software engineering capability, applications experience, programming language experience, ...
- Project attributes: use of software tools, application of software engineering methods, ...
Intermediate COCOMO

- The 15 sub-attributes are each rated from "very low" to "extra high" with six discrete choices
- Effort multipliers are empirically derived and the EAF is the product of the multipliers

<table>
<thead>
<tr>
<th>Overiload</th>
<th>Low</th>
<th>Medium</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product attributes</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Required software capability</td>
<td>1.10</td>
<td>1.00</td>
<td>0.90</td>
<td>0.80</td>
<td>0.70</td>
</tr>
<tr>
<td>Complexity of the product</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Baseline attributes</td>
<td>1.10</td>
<td>1.00</td>
<td>0.90</td>
<td>0.70</td>
<td>0.50</td>
</tr>
<tr>
<td>Projected schedule</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Projected development costs</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Available personnel</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Elapsed development time</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Predicted development costs</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Predicted development schedule</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Predicted staff size</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Use of software tools</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Evaluation of software engineering methods</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
<tr>
<td>Required development environment</td>
<td>0.75</td>
<td>0.65</td>
<td>0.55</td>
<td>0.45</td>
<td>0.35</td>
</tr>
</tbody>
</table>

E = a(KLOC)^b \times \text{EAF}

And similarly for development time and people counts

There is a separate table for parameters a and b across organic, semi-detached, embedded for Intermediate COCOMO

Detailed COCOMO & COCOMO II

- Detailed COCOMO also accounts for the influence of individual project phases
- COCOMO II was developed and released in 1997, aimed at (then) modern software projects
  - Newly tuned parameters
  - Accounted for move from mainframes to desktops, from batch to interface computation, to code reuse, etc.

1981 Boehm book also discusses

- Multiple-goal decision analysis
- Most optimization theory assumes that there is a single objective function to maximize
- Models like this one account for multiple goals that must be balanced in a definable manner
- Risk analysis
- Foundation for his later work in the spiral model
- And more...
"The core competency of software engineers is in making technical software product and process design decisions. Today, however, there is a ‘disconnect’ between the decision criteria that tend to guide software engineers and the value creation criteria of organizations in which software is developed. It is not that technical criteria, such as information hiding architecture, documentation standards, software reuse, and the need for mathematical precision, are wrong. On average, they are enormously better than no sound criteria. However, software engineers are usually not involved in or often do not understand enterprise-level value creation objectives. The connections between technical parameters and value creation are understood vaguely, if at all. There is rarely any real measurement or analysis of how software engineering investments contribute to value creation. And senior management often does not understand success criteria for software development or how investments at the technical level can contribute fundamentally to value creation. As a result, technical criteria tend to be applied in ways that in general are not connected to, and are thus usually not optimal for, value creation."

Thinking about value

- Decision theory (or utility theory) defines a framework for decisions under uncertainty, depending on the risk characteristics of decision makers
- This is closely related to (again) multi-objective decision-making
- Classical corporate finance uses net present value (NPV) as an investment decision criterion and computes it by discounted cash flow analysis (DCF) – can’t make a business case without these

NPV example from Wikipedia

- A corporation must decide whether to introduce a new product line. The new product will have startup costs, operational costs, and incoming cash flows over six years. This project will have an immediate (t=0) cash outflow of $100,000 (which might include machinery, and employee training costs). Other cash outflows for years 1-6 are expected to be $5,000 per year. Cash inflows are expected to be $30,000 each for years 1-6. All cash flows are after-tax, and there are no cash flows expected after year 6. The required rate of return is 10%.
Con’t

- The table shows the present value (PV) for each year.
- The NPV is the sum of the PVs.
- In this case, it’s $8,881.52.
- A positive NPV means it would be better to invest in the project than to do nothing — but there might be other opportunities with higher NPV.

<table>
<thead>
<tr>
<th>Year</th>
<th>Cashflow</th>
<th>Present Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T=0</td>
<td>$-100,000</td>
<td>$-100,000</td>
</tr>
<tr>
<td>T=1</td>
<td>$30,000 - $5,000</td>
<td>$(1 + 0.10)^3$</td>
</tr>
<tr>
<td>T=2</td>
<td>$30,000 - $5,000</td>
<td>$(1 + 0.10)^3$</td>
</tr>
<tr>
<td>T=3</td>
<td>$30,000 - $5,000</td>
<td>$(1 + 0.10)^3$</td>
</tr>
<tr>
<td>T=4</td>
<td>$30,000 - $5,000</td>
<td>$(1 + 0.10)^4$</td>
</tr>
<tr>
<td>T=5</td>
<td>$30,000 - $5,000</td>
<td>$(1 + 0.10)^5$</td>
</tr>
<tr>
<td>T=6</td>
<td>$30,000 - $5,000</td>
<td>$(1 + 0.10)^6$</td>
</tr>
</tbody>
</table>

Real options

- DCF/NPV treats assets as passively held — not actively managed.
- But projects are (or can be 😊) actively managed.
  - Management usually has the flexibility to make changes to real investments in light of new information. (e.g., to abandon a project, enter a new market, etc.)
- The key idea of real options is to treat such flexibility as an option, and to (in some cases) price them using techniques related to those for financial options.

Baldwin and Clark (2000)

- Baldwin and Clark view Parnas’ information hiding modules as creating options.
- They value these and develop a theory of how modularity in design influenced the evolution of the industry structure for computers over the last forty years.
- Non-modular systems must be kept or replaced as a whole.
- A system of independent modules can be kept or replaced (largely) individually based on judgments of improvement (or not).
- Modularity provides a portfolio of options vs. an option on a portfolio.

DSMs: design structure matrices

- The parameters are A, B, and C.
- The X in row B, column A means that good choice for B depends on the choice made for A.
- Parameters requiring mutual consistency are interdependent, resulting in symmetric marks: (B,C) and (C,B).
- When one parameter choice must precede another, the parameters are said to be hierarchically dependent: (B,A).
- Independent parameters can be changed without coordination.

Material from Sullivan, Griswold, Cai, Hallen. The structure and value of modularity in software design. ESEC/FSE 2001
Splitting

DSMs may not show largely independent designs

In these cases, one approach is to apply splitting

Break a dependence with a new parameter that constrains the values of the original parameters – this means, in part, that they depend on it

Fix the value of the new parameter so that the original parameters to be changed independently as long as they are only changed in ways consistent with the new constraint

For example, introduce a new interface (I, in the below example)

---

A module creates an opportunity

- to invest in $k$ experiments to create candidate replacements,
- each at a cost related to the complexity of the module
- if any of the results are better than the existing choice, to substitute in the best of them
- at a cost that related to the visibility of the module to other modules in the system

---

The option value of each module is the value at the peak

Sum the module NOVs

- 0.26 for the strawman design
- 1.56 for the information-hiding
Status

- The basic idea seems to make sense to many people.
- One of the core problems is the notion of how to tune the model parameters.
  - Financial markets set parameters based primarily on scads of historic data.
  - COCOMO set parameters based on careful studies of a reasonably large set of reasonably similar software projects.
  - Tuning parameters for modularity seems more complicated.

Governance of Software Development

- Clay Williams, IBM Research
- Slides directly taken from an NSF workshop presentation.

McConnell’s cone of uncertainty

ICSE 2009 keynote
Governance of Software Development Strategic Initiative

- Goal: Develop the science and technology that enables the Rational software delivery platform to provide support for governing the business of software development.

Development Governance

- Value and Risk Management
- Organizational Design and Collaboration

Tempo - Overview

- Problem Statement
  - When project teams commit to a schedule, they are placing a bet. It would be extremely valuable for them to know the odds of winning.

- Approach
  - Capture "bottom-up" predictions regarding the time necessary to complete each task in a work breakdown.
  - Rather than discrete predictions, capture triangular distribution that reflects the fact predictions are random variables.
  - Develop optimal scheduling approaches that rapidly reduce schedule risk in the project.
  - Surface schedule risks to allow teams to better manage scheduling issues.

- What is hard?
  - Providing a tool that is easy to use and supportive of "what-if" risk mitigation analysis requires addressing subtle and difficult usability issues.
  - The variety of optimizations and analyses require significant mathematical skill.

Tempo in Rational Team Concert

Architectural and Social Governance of Software Development

- Research Goals
  - Exploit / expand the role architectures play as "boundary objects" spanning multiple domains of discourse.
  - Develop techniques for exploring key structural and behavioral properties of architectures (software, IT, and EA), the socio-technical dynamics of the teams producing and consuming them, and how these two areas can be aligned and engender communication beyond the technical domain.
  - Develop / extend architectural approaches to support business decisions and value management.
  - Understand the interplay across the value / architectural / socio-technical domains.

- Collaborations
  - CMU (Jim Herbsleb)
  - Harvard Business School (Carliss Baldwin) - pending
  - Virginia (Kevin Sullivan)
The long-term goal of software engineering economics is to help everybody make more sensible decisions:
- Technical decisions
- Business decisions
- Project management decisions

Not one of these is primary with the others secondary — but that is how we each seem to treat the others.

Better understanding the links among them is crucial; the models may give us opportunities to better understand these links.

I am always scared that quantification tends to lead to a focus on the quantities, and there is often a disconnect between the quantities we can measure and want we want to do.