

Workshop Preparation Guidelines

NSF Software Research Strategy Workshop

General Guidelines

Workshop Objective and Focus. We've convened a group of largely software engineering researchers, and couched some of the issues in terms of software engineering research. However I think our main objective and focus should be to identify good research ideas and strategies that address the software problems identified in the PITAC report (fragility, unpredictability, etc.). Whether these ideas and strategies fit some particular definition of "software engineering" is, I think, a topic for some other time.

Clearly, we don't want to stray too far from our core competence. However, there are some software engineering borderline ideas suggested so far, e.g., "exploring the applicability of real-option theory and related economic theories to software decisions," and "developing synthetic experiences that educate people to become better software customers," for which I think it would be unproductive to spend too much time wrangling about a definition of software engineering that adjudicates where these would fall.

The particular focus areas for the workshop at this point are: setting realistic expectations for software engineering research; preparing success stories on significant previous software engineering research payoffs; identifying critical success factors for software engineering research strategies; and identifying software research Grand Challenge problems that could stimulate new research ideas and high-payoff solution approaches (these and any of the guidelines are suitable topics for suggestions and discussion).

Workshop Outcome. The primary outcome of the Workshop needs to be a briefing and executive summary that provide NSF and other organizations with useful and convincing research ideas and strategies for addressing the PITAC software problems. While achieving this primary outcome, we have some great opportunities to create a shared vision and strategies for galvanizing the software engineering research community, but these won't be effective if we can't achieve the primary outcome.

A draft outline of the briefing is provided below:

- Workshop Objectives
- Workshop context (PITAC, IT², Basili report, etc.)
- Nature of the problem space and solution space
 - Realistic expectations
 - Success stories (evidence of research payoffs)
 - Pitfalls (?)
- Research strategies
 - Lessons learned from the success stories and the pitfalls
- Grand Challenges
- Conclusions and Recommendations

Issues for Email Discussion Before the Workshop

1. Is it realistic to expect “silver bullet” solutions to the entire software problem (for software in general? within restricted domains?) Can we rest our case on Brooks’ paper here? Are there other points that are important to make?
2. Is it realistic to expect instantly transferable software engineering research results? How does the 15-20 year software technology maturation time in the Redwine-Riddle ICSE-8 (1985) paper (highly relevant but hard to locate these days; we’ll put a scanned copy on the Web site) relate to the Internet-time phenomena we see now? Are there discriminators that affect transition time (e.g., whether or not you have to modify your behavior to adopt the technology)?
3. What is realistic to expect from software practice without significant software engineering research? We are not the ideal group to address this question. But if some of you have industrial experience via employment, consulting, etc., that can help illuminate inhibitors to industry solutions research (short-term focus, risk aversion, more rewards for solving customers’ problems than developers’ problems, etc.), it would be useful to contribute short example experiences.
4. What are some well-founded software engineering research success stories? Some examples compiled by Lori Clarke, Lee Osterweil, Dewayne Perry, and Dick Taylor in a recent NSF briefing are: Process, Configuration Management/Version control, Internet Protocols, Integrated Development Environments, Testing Tools and Technology, Inspections, Metrics, High Reliability, Requirements, and Design (we’ll also put these briefing charts on the Web site).

I think it would be valuable to try to capture Software Engineering Research Success Stories in some sort of standard format. Here’s a candidate format; I’ve illustrated it below using Requirements Technology.

Software Engineering Research Success Story

- Name of capability:
- Description of capability:
- Software engineering research foundations:
- Examples of successful transition:
- Evidence of significant payoffs:
- Representative references:

I’d encourage everybody to contribute and discuss instances of these.

5. What are frequent pitfalls and critical success factors for software research strategies? The critical success factors tend to be reflections of the success stories and the pitfalls: e.g., don’t overinflate expectations; don’t ignore commercial technology;

don't ignore empirical data; don't put all your eggs in one basket; skate to where the puck is going rather than to where it is or where it's been (Wayne Gretzky); develop scientific underpinnings; etc.

I think the best way to contribute these is via a short title and short explanation of each. Another topic for discussion is whether emphasizing pitfalls is being too negative.

6. What are candidate software engineering "Grand Challenge" problems that could stimulate new research ideas and high-payoff solution approaches? Some successful examples from the 1992 HPCC Report were: Forecasting Severe Weather Events, Cancer Genes, Predicting New Superconductors, Air Pollution, and Aerospace Vehicle Design. These were accompanied by a 1-page summary of how HPCC technology could expedite solution of the problem, and an impressive color illustration.

For software, doing color pictures is generally harder. Again, I'd suggest trying to capture these in a common form. The form Dick Taylor used in his 8/3/99 email looks good. I've also appended one of his examples below.

Software Engineering Research Grand Challenge Problem

Grand Challenge:

Situation Today:

Hypothesis:

What This Will Require (Underlying Science):

How Progress Can Be Measured:

Here are some candidate software engineering Grand Challenge problems that have been suggested to date:

- Keeping one billion lines of code under the intellectual control of a single individual;
- Ultra-light weight, customer-understandable formal methods;
- Automatic Programmer. Devise a specification language or user interface that: (a) makes it easy for people to express designs; (b) computers can compile, and (c) can describe all applications (is complete)
- 10x increase in functions provided per line of development "code" every 10 years;
- Enable N people to collaborate remotely on a project with N times the effectiveness of one person;

- Giving a person in one part of the world the ability to automatically learn about and react to an event occurring in another part of the world;
- Self-healing, context-aware software;
- Making software change efforts proportional to the size of the change, not the size of the product;
- “Portable contexts” for (people, software components) that enable other (people, software components) to more effectively collaborate with them;
- Integrated software/hardware/system definition and development;
- Synthetic experiences that educate people to become better software engineers or customers;
- The equivalent of air bags or seat belts for user-programmers;
- Applying economic theory (value of information, real options theory) to software engineering decisions;
- Early identification of software projects that should be terminated;
- Experimental science for software engineering at a level equivalent to that of both the physical and behavioral sciences.

In addition, here’s Neno’s somewhat overlapping but categorized list of candidate Grand Challenges:

Cost and productivity

- Better, cheaper, faster -- pick all three!
- 10x increase in productivity every 10 years
- Effectiveness (N developers) = N * Effectiveness(1 developer)
- Accurate cost estimation models

Distributed development

- Automatic event notification
- Social context of collaboration and cooperation
- Eliminate the need for programmers

Alternative approaches to design

- Minimal commitment mode of design
- Make architectural design knowledge available to all
- Multi-level models (automatically) refined into software

Correctness and reliability

- All systems work correctly the first time
- Software testing no longer required
- Establishing experimental science for software engineering

Relationship with hardware

- Integrate software as successfully as hardware
- Automatic reconfiguration to optimize with respect to the hardware
- Build software to direct hardware reconfiguration

Dealing with complexity

- Domain-specific software engineering
- Generalization from domain experience
- Reuse across services and domains

Evolution flexibility

- System upgrades invisible
- \$0 cost to adapt
- Self evolving/healing/aware software

Again, please try to elaborate candidate Grand Challenges into the common form above, discuss their suitability as Grand Challenges and how to organize them, or suggest others.

7. Suggest or discuss any other topics for the Workshop objectives and outcome; agenda; or preparation activities.

Software Engineering Research Success Story

Name: Requirements Technology

Description: Tools and techniques for determining, representing, analyzing, and managing software and system requirements.

Software Engineering Research Foundations: These are best captured in a landmark set of papers at the Second International Conference on Software Engineering in 1976. Most of the papers are published in the January 1977 issue of the IEEE Transactions on Software Engineering.

- The Teichrow PSL/PSA system, developed at the University of Michigan.
- The TRW SREM system, which drew on PSL/PSA and other software research such as the UCLA graph model of computation, the Lecarme-Bochmann compiler writing system, and the Allen-Cocke dataflow analysis techniques.
- The SofTech SADT system, which also drew on PSL/PSA, and on Ross' extensive prior research at MIT on AED and related concepts.
- Empirical studies by Bell and Thayer on the nature of requirements defects, also drawing on earlier empirical results from Bell Labs, GTE, IBM, and TRW on the payoffs due to early defect removal.

Concurrent and subsequent research on formal specifications and the Parnas A-7 project have also had significant impact.

Examples of successful transition: PSL/PSA was adopted by AT&T, Chase Manhattan Bank, Mobil Oil, TRW, and many other companies. As CADSAT, it benefited from significant Air Force productization support and use.

SREM, with significant Army productization support, evolved into Ascent Logic's RDD-100 tool.

SADT has also had success as a commercial requirements tool, and especially via its evolution into the IDEF series of requirements tools sponsored by the DoD ManTech program.

Numerous other commercial requirements tools such as DOORS, Requisite Pro, RTM, and Software Through Pictures have also drawn on this technology base.

Evidence of significant payoffs: Davis' book on Software Requirements has considerable evidence of payoffs. Grady's book on metrics experience at Hewlett Packard has similar evidence. A number of other value-chain and cost-of-quality analyses indicate that the major payoffs are in rework avoidance via early defect prevention and removal. As a rough approximation, requirements defects on a weak-technology project typically account for 40% of rework costs, which are typically 40-50% of its total software development costs. This creates a potential technology savings of 16-20% per project; conservatively, requirements technology currently achieves about half of this. At the current rate of U.S. software development costs of roughly \$800 billion/year, this is an annual savings of \$64-80 billion per year.

Representative references:

Alford, M., "A Requirements Engineering Methodology for Real-Time Processing Requirements," IEEE-TSE, January 1977.

Bell, T., D. Bixler, and M. Dyer, "An Extendable Approach to Computer-Aided Software Requirements Engineering," IEEE-TSE, January 1977.

Bell, T., and T. Thayer, "An Extendable Approach to Computer-Aided Software Requirements Engineering," Proceedings, ICSE2, 1976, pp. 61-68.

Davis, A., Software Requirements, Prentice Hall, 1993.

Grady, R., Practical Software Metrics for Project Management and Process Improvement, Prentice Hall, 1992.

Ross, D., and K. Schoman, "Structured Analysis for Requirements Definition," IEEE-TSE, January 1977.

Teichroew, D., and E. A. Hershey., "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis," IEEE-TSE, January 1977.

Software Engineering Research Grand Challenge Problem

Grand Challenge: 10x increase in functions provided per line of development "code" every 10 years.

Situation Today: This increase has taken place every twenty years, thus we seek to double the rate of this advance.

Hypothesis: Development of reuse techniques, formal specifications, software architectures, and middleware has increased dramatically within the last decade --- but their combination and application in industrial projects has yet to be seen. The opportunity is thus present for these results to be combined and transitioned to widespread practice.

What this will require (Underlying science):

- o development of effective techniques for measuring functionality delivered
- o new development technologies and tools

How progress can be measured:

- o at a minimum metrics such as object code size or function points can be correlated with the size of the lowest level of specification that human developers work with (e.g., programming language code, architecture descriptions, or abstract specifications, as appropriate).