This workshop study was commissioned by the National Science Foundation (NSF) to obtain a clearer understanding of candidate research strategies in software engineering, with the objective of gaining intellectual control of software development. For this study, the term “development” was defined to include software evolution and legacy software adaptation.

It was funded by NSF grant #CCR-9985707 to the University of Southern California (USC). The funding covered only the Workshop operational expenses and the participants’ travel expenses. Its Principal Investigator was Dr. Barry Boehm of the USC Computer Sciences Department. Its primary sponsors were:

Dr. Ruzena Bajcsy, NSF Associate Director for Computer and Information Science and Engineering;

Dr. Michael Evangelist, Director, Computer and Communication Sciences Division;

Dr. Frank Anger, Manager, Software Engineering and Languages Program
The workshop convened many of the leaders in the software engineering field. Participants were selected to balance research areas; gender; senior and junior researchers; and academia and industry.

The Program Committee members indicated by asterisks helped organize the August 24-25, 1999 Workshop session at USC in Los Angeles, and participated as speakers in the Workshop out briefing September 27, 1999 at NSF in Washington DC. The remote participants contributed to the discussions via email and an electronic bulletin board organized by George Heineman.
Outline: Workshop Results

- Scope and Context
- Nature and Role of Software Engineering
- Software Engineering Research Strategies
- Conclusions

Chart 3.

This overview briefing covers the overall scope and context of the Workshop, the Workshop approach, and the overall findings. Additional briefing charts cover the detailed results.
The study was motivated by the report by the President's Information Technology Advisory Committee (PITAC) on "Information Technology Research: Investing in Our Future" [PITAC, 1999]. Among other items, it identified computer software as both a major enabling factor and a major limiting factor in achieving the potential of information technology. It concluded that Federal IT research was seriously underfunded, and identified software research as a top-priority area for increased emphasis.

This led to a significant proposed increase in IT research funding starting with the President's proposed Fiscal year 2000 budget. In response, a number of Federal agencies sponsoring IT research joined together to produce a strategic plan for the research initiative, initially called Information Technology for the Twenty-First Century (IT2), and subsequently renamed the IT Research initiative [NCO, 1999].

In October 1998, NSF sponsored a Software Research Program Workshop to identify research opportunities for addressing the software fragility and unpredictability problems identified in the 1998 draft of the PITAC Report. It focused on achieving a goal of achieving "no surprise" software via thorough understanding of software phenomenology via empirical research; improvements in software product and process capabilities; and mechanisms facilitating software technology transition [Basili et al., 1999].

A number of questions were raised about whether this report was recommending a "software engineering" vs "software" research agenda and how it addressed other software research areas such as operating systems, networking, database, artificial intelligence, and database software.

The workshop summarized here was convened to clarify the nature and role of software engineering (SE) research and technology in addressing the PITAC software challenges, and to identify appropriate SE research strategies within the framework of overall software and IT research.
If an organization wants a good information technology (IT) based system, it needs a synergetic relationship between the IT components (in black) which comprise the delivered system, and the software engineering (SE) elements (in red) which guide the definition, development, component selection, integration, and verification of the system.

If we want to go from good to great IT systems, it is not sufficient to improve the SE elements without attention to improving the IT components. It is equally true that just focusing on great IT components without attention to complementary SE practices generally does not succeed.

A good example of the latter is the AESOP project experience documented in [Garlan et al., 1995]. The project assumed that a small number of advanced IT components (a user interface generator, an object-oriented DBMS, and two middleware components) could be integrated quickly and cheaply (6 months, 2 people) via a “simple matter of programming” by very bright software engineering researchers. The result: a schedule overrun by a factor of 4; an effort overrun by a factor of 5; and slow, unresponsive system performance. Similar approaches on large-scale systems generally experience even worse results [Standish, 1995], but are generally not documented.

The saving grace of the AESOP experience was that the failure was well-analyzed by the SE researchers. The analysis identified “architectural mismatch” as the key success inhibitor, leading to a rich experience-based SE research program to address and cope with the sources of architectural mismatch [Shaw-Garlan, 1996].

Further discussion of the AESOP experience and associated software architecture research issues and opportunities are found in Nenad Medvidovic’s presentation below.
Future Software Trends

- Increased complexity
  - Everything connected
  - Opportunities for chaos (agents)
  - Systems of systems
- Decreased control of content
  - Infrastructure
  - COTS components
- Faster change
  - Time-to-market pressures
  - Marry in haste; no leisure to repent
  - Adapt or die (e-commerce)
- Fantastic opportunities
  - Personal, corporate, national, global

Chart 6.

Achieving great, no-surprise IT systems via stronger and better-integrated IT components and SE capabilities would be a tremendous challenge even if the nature of software and IT were to hold still for a few years. But these fields are rapidly changing in ways that make the situation ever more challenging. The Web and the Internet connect everything with everything else. Autonomous agents making deals in cyberspace create many opportunities for chaos. Systems of systems, networks of networks, and agents of agents create huge intellectual control problems.

Further, the economics of software componentry leave system developers with no choice but to incorporate large commercial-off-the-shelf (COTS) components into their systems. Developers have no way of knowing what is inside these COTS components, and they have no control over their directions of evolution.

The PITAC Report accurately states (page 8) that “The IT industry expends the bulk of its resources, both financial and human, in rapidly bringing products to market.” And the dizzying pace of this change continues to increase. Software architecture and COTS decisions are made in great haste. If you marry your IT architecture in haste, you no longer even have the opportunity to repent at leisure. Commercial companies with minimal electronic commerce capabilities must somehow adapt to e-commerce or die.

Of course, these trends also make it a time of fantastic opportunity. The PITAC Report is “right on” in emphasizing that IT offers us the potential to significantly improve our overall quality of life by transforming the ways we learn, work, communicate, and carry out commerce, health care, research, and government.
Chart 7.

Having clarified in Chart 5 the nature and role of software engineering, and its synergetic role with IT components research in realizing increasingly better IT systems, we turn to the problem of characterizing software engineering elements and their individual roles in realizing various classes of future IT systems.

Before continuing, however, we would like to clarify an oversimplification in Chart 5. This is that the advanced tools and environments developed in software engineering research also contribute to the store of “great IT components,” and represent another area of synergy with the IT components research areas. Richard Taylor’s presentation below on “Connectivity and Information Access” provides some recent examples.
Chart 8.

The Workshop's approach to the characterization of SE research elements began by reviewing and discussing another excellent source of overall software Grand Challenge Problems [Gray, 1999]. With the aid of these ideas, the Workshop participants defined two SE-intensive challenge areas involved in achieving the quality of life improvements envisioned in the PITAC Report. These involved the harnessing of future IT in "Empowering people and groups," and in "Weaving a new information fabric" that is much more reliable, supple, and adaptable than our current information fabric.

Within these two challenge areas, the Workshop identified four specific example applications, and formed breakout groups to characterize their potential future IT/SE-enabled capabilities, and the relative importance of individual SE research areas in achieving the capabilities. These were then aggregated into a cross-impact matrix which clarifies the relationships, and leads to a number of conclusions about the nature of SE research strategies.
Empowering People and Groups: Workshop Examples

- User Programming
- Empowered Teams
- Lifelong Learning
- Embedded Medical Systems

Chart 9.

In the first challenge area of Empowering People and Groups, the four example applications and their breakout group participants were:

- User Programming: Bob Balzer (USC-ISI); Dewayne Perry (Lucent); Mary Shaw (CMU)
- Empowered Teams: Lori Clarke (Umass); Susan Graham (UC Berkeley); Beki Grinter (Lucent); Richard Taylor (UC Irvine)
- Lifelong Learning: George Heineman (WPI); Stuart Zweber (Ohio State U.)
- Embedded Medical Systems: Nancy Leveson (MIT); Victoria Stavridou (SRI).
Weaving the New Information Fabric: Workshop Examples

• Crisis Management
• Air Traffic Control
• On-Demand Organizations
• Medical Informatics

Chart 10.

In the second challenge area of weaving the New Information Fabric, the four example applications and their breakout group participants were:

• Crisis Management: Frances Allen (IBM); Peter Freeman (Ga. Tech); William Scharlis (CMU)
• Air Traffic Control: Victor Basili (U Maryland); John Goodenough (CMU-SEI); John Knight (U Virginia)
• Net-Centric Business: Richard De Millo (Telcordia); Nenad Medvidovic (USC); Kevin Sullivan (U Virginia)
• Medical Informatics: Jim Horning (Intertrust); Leon Osterweil (U Mass)
As a cross-check on the application area analyses, three Program Committee members elaborated a selection of key software engineering technology areas and their most significant research challenges in the light of the technology needs identified in the application areas. Summaries of these are also provided in further presentations below:

- Software Architectures: Nenad Medvidovic (USC)
- Connectivity and Information Access: Richard Taylor (UC Irvine)
- Modeling and Analysis: Victor Basili (U Maryland)
Chart 12.

The first eight columns in chart 12 summarize the degree of dependence of each of the eight example applications on the primary software engineering technologies identified in the application analyses. A very large dot (●) indicates that making significant progress in the technology area is essential to making significant progress in the application area. A large dot (★) indicates a strong dependence; a small dot (●) indicates a moderate dependence; and no dot indicates no dependence.

As discussed in Chart 7, the Product Technologies shown in chart 12 represent a shared challenge for both the software engineering research and the IT components research shown in Chart 5.

As an example column of the matrix, consider the first column, "User Programming." The ratings for degree of dependence of successful user programming on the various software engineering technologies at the left of chart 12 were derived from the User Programming example application analysis summarized in Dewayne Perry’s set of charts below.
In order to achieve safe and effective user programming in the future, there is an essential dependence on achieving significant progress in safe and effective composition of components and applications; on test and verification that the users programs have no serious faults or adverse side effects; on supportive human-computer interface (HCI) technology; on user domain componentry for composing user applications; on high-assurance technologies for ensuring that user programs do not compromise reliability, privacy, availability, safety, mission performance, or organizational viability; on change resilience to ensure that user modifications can be done quickly and with high assurance; and on ensuring that each technology integrates well with the others (e.g., that user domain componentry evolves in ways consistent with the change resilience assumptions).

The other software engineering technologies are less essential to successful user programming, primarily because the programs and projects are relatively small. Massive scalability is no needed at all, and process and economics modeling contribute in only moderate way. The strong dependencies for the other technologies primarily address their need in establishing sound support frameworks and environments for user programming, e.g., system definition; architecture; connectivity and information access; and information distribution and management.

The final four columns in Chart 12 indicate the relative degree of dependence of each software engineering technology on four primary underlying science areas: computer science, domain sciences (e.g., physics for physical systems), behavioral sciences, and economics. This selection of science areas is not complete: one could clearly add mathematics and physiology, for example.

Thus, for example, if a set of stakeholders were trying to negotiate the definition of the best air traffic control system that can be built within a given budget and schedule, the system definition support capabilities would depend in an essential way on the knowledge of computer science (e.g., on the performance of algorithms); on domain sciences (e.g., on the computations needed for the prediction of clear air turbulence); on behavioral sciences (e.g., for both stakeholder requirements negotiations and air traffic controller group performance); and on economics (e.g., for defining appropriate success models for the system and performing cost-benefit analyses of various alternatives).

Determining the most appropriate logical and physical IT architecture for the air traffic control system would involve a strong dependence on all four underlying sciences, but the dependence on getting the best information structures would be considerably more essential for computer sciences than for the other sciences.
Conclusions: SE Research Needs

- Major needs for further SE science and technology (S&T)
  - Process, product, quality of service, modeling and analysis
  - S&T integration across areas
- SE science base requires more than computer science
  - Need integration with domain sciences, behavioral sciences, economics...
  - Need both specialist and interdisciplinary advances
- There is no single silver bullet for success
  - Major applications require many technologies
  - Need integrated SE/IT research programs

One can draw a number of conclusions from the larger and smaller dots in Chart 12 showing the relative dependence of the eight example future application challenges on individual software engineering technologies. The most significant conclusions with respect to software engineering research needs are the following:

*The needs are significant.* Each of the eight example challenge applications exhibited essential or strong dependencies on improved capabilities in most of the software engineering technology areas. Having integrated, mutually reinforcing technology elements was essential for all the challenge applications. As discussed on Chart 5, having either good product or process technology without the other is unlikely to produce a good system.

*The needs are interdisciplinary.* As discussed above, system definition technology, architecture technology, and the other SE technologies require more than traditional computer science to ensure successful IT applications. This does not imply that single-discipline research is not important. It does imply that the "body of knowledge" required for successful software engineering includes considerably more than computer science.

*There is no single silver bullet for success.* The wide distribution of larger dots in Chart 12 corroborates the [Brooks, 1987] thesis that the "essential" vs. the "accidental" problems in software engineering require more than a single silver-bullet solution. For research strategies, this implies that a balanced portfolio of research investments and an emphasis on integration of software engineering and information technology solutions via experimental application are most likely to show progress toward addressing the future trends and challenges discussed in Chart 6.
Conclusions: SE Research Strategies

- Future software trends create integrated SE/IT challenges
  - Increased complexity
  - Decreased control of content
  - Faster change
  - Fantastic opportunities
- Weaving new information fabric provides vision of solution
- Need integrated SE/IT research program to realize the vision

Chart 14

In conclusion, the future trends discussed in Chart 6 offer us not just challenges but fantastic opportunities. The eight application problems summarized above and elaborated in individual presentations below provide a vision of the nature of their potential solution: weaving a new information fabric that is more reliable, supple, and adaptable than our current information fabric. Realizing such a vision will require the best collaborative efforts possible among software engineering researchers, IT component researchers, and IT development practitioners.

Some backup charts elaborate on further research strategy aspects.
Backup charts

- SE Research Strategy Guidelines
- Achieving No-Surprise Software
- SE Research and Development Progress Metrics

Chart 15

Some backup charts address:

- Further research strategy guidelines (Charts 16-18)
- Relation of the research strategy to the achievement of the goal of "no-surprise software" in [Basili, 1999] (Charts 19-20).
- Candidate examples of progress metrics for software research and development (Charts 21-22).
Investment Strategy Guidelines: SE Research

- Increase level of investment in individual SE/IT research projects
  - Some critical success factors on next chart
- Significant level of investment in empirical studies of software phenomena
- Increasing levels of investment in experimental research
  - Initially medium-scale trials
  - Where feasible, followed by industry co-funded larger trials

Chart 16

The study concluded that the level of investment for individual software engineering and information technology research efforts should be appreciably increased. In addition, a significant level of investment should be devoted to empirical studies of software phenomena, to provide better understanding of the areas most needing solutions and how they interact.

More investment is also needed in experimental research to test and refine research hypotheses and to better address the scalability of research solutions. The level of such investments should grow with our understanding of how to carry them out effectively.
Critical Success Factors for SE Research Programs

- Emphasize scientific foundations
  - Clear hypotheses; careful measurements; repeatability
  - Evaluated w.r.t. alternatives and domain of applicability
- Broaden empirical understanding of software phenomenology
  - Enables focus on high-leverage problems and solutions
- Skate to where the puck is going (Gretzky)
  - Anticipate and address future problems
- Maintain a balanced research portfolio
  - Evolution/revolution; basic/applied; theory/systems
- Expand horizons via Grand Challenges
- Stimulate “out of the box” ideas
  - New metaphors: biology, sociology, economics
- Stimulate university-industry collaboration; transition into practice

Chart 17

Most of the critical success factors on this chart are fairly self-explanatory. Here are a few elaborations.

- **Skate to where the puck is going.** Workshop participants felt that considerable research is focused too much on problems of the past. Examples of future trends needing more research into their intellectual control are heterogeneous distributed systems, dynamically changing software structures, and interactions among autonomous agents.

- **New metaphors.** One approach to problems of the future is to look for new perspectives or metaphors for intellectual control: e.g., biologically self-testing or self-adaptive software systems; socioeconomic employment of software goal and reward structures.

- **Technology transition.** This requires special attention in software engineering, particularly due to factors elaborated on the next chart.
Software Engineering Technology Transition Challenges

- Adoption requires behavioral change
- Payoffs take a long time to demonstrate
  - And are hard to trace back to particular technology insertions
- Marketplace often makes “fixing it later” more attractive than “doing it right the first time”
  - “The IT industry expends the bulk of its resources, both financial and human, on rapidly bringing products to market.” - PITAC Report, p.8
- Strong coupling among technologies, processes, acquisition practices, cultures
- Rapidly evolving commercial technology
- Slowly evolving Government acquisition practices
- Risk-averse program managers
- Leaky intellectual property

Chart 18

Software engineering technology is harder to transition than most other technologies, due primarily to the following phenomena:

- **Adoption involves behavioral change and deferred gratification.** Plugging in a faster chip or algorithm can be done without changing one’s project practices, and the effect on performance can be seen immediately. Changing one’s software processes or adopting software product line practices requires people to change their behavior on projects, often for payoffs which are only seen much later and are hard to trace definitively.

- **Other disincentives to embracing software process change are** “I don’t have time,” “It would add another element of risk,” “It’s likely to be obsolete in two years,” or “It doesn’t give me a sustainable competitive advantage.”

- **Still, the IT industry is looking for better software methods.** Collaborative university-industry research and experimentation is a good way to expedite transition. Several good suggested mechanisms are in Section 5 of [Basili, 1999].
Expected Payoff: Practical Achievement of No-Surprise Software

- Expanding domain of applications with well-understood solution approaches
  - Feasible to complete project with no surprises (overruns, user mismatches, performance/reliability problems)
  - Mix of cycle time, cost, functionality, levels of service matched to stakeholder needs

- Clear demarcation of no-surprise boundary
  - Identification of likely problems outside boundary
  - No surprise that problems will be encountered

Chart 19

One effect of relating software research results to key challenge applications will be to provide a higher level of understanding of which processes, architectural styles, and IT components best fit such applications. This provides the foundation for no-surprise projects in another applications domain, and the ability to reason about what level of capability can be achieved within a given time or budget.

Keeping track of progress toward this objective will make it clearer which challenge problems are outside the no-surprise boundary, providing stakeholders with more realistic expectations about encountering problems. It will also enable them to consider alternative strategies for bounding risk, such as using delivery time or cost as one’s independent variable, and desired features as the dependent variable.
Here is an example chart showing a basic level of understanding of how various specification methods fit which types of applications. An application with very stable requirements and a very high need for fault-freedom is a good candidate for using mathematically precise formal specifications. If the application requirements change rapidly, the cost and schedule of reworking the formal specifications and verifications may outweigh their benefit, particularly if the application's required fault-freedom is relatively low. A major challenge is to quantify the measurement scales, but even in its qualitative form, this chart has been useful in guiding industrial decisions.
The workshop also discussed the formulation of appropriate metrics for software development progress, and their use in evaluating the effects of advances in software engineering research. The main conclusions were that the task is difficult, but that this no reason not to do better.

The main difficulty arises from the fact that software projects in practice are effectively unrepeatable. One has either differences among team skills and team dynamics, or one has learning curve effects on similar products developed by the same team. And, increasingly, rapid changes in the IT marketplace and infrastructure continually change some of the rules of the software development game from one project to the next.

This changing nature of software development means that most traditional measures of software productivity are increasingly irrelevant. A good example is the metric of new source lines of code (SLOC) produced per person-day on a large project. The value of this metric continues to hover around 10, giving an impression of no progress. However if one looks at Chart 21, reflecting several decades of experience at Bell Laboratories summarized in [Bernstein, 1997], the number of executing lines of machine code generated by a line of source code has increased by roughly an order of magnitude every 20 years.

**Chart 21**

The workshop also discussed the formulation of appropriate metrics for software development progress, and their use in evaluating the effects of advances in software engineering research. The main conclusions were that the task is difficult, but that this no reason not to do better.

The main difficulty arises from the fact that software projects in practice are effectively unrepeatable. One has either differences among team skills and team dynamics, or one has learning curve effects on similar products developed by the same team. And, increasingly, rapid changes in the IT marketplace and infrastructure continually change some of the rules of the software development game from one project to the next.

This changing nature of software development means that most traditional measures of software productivity are increasingly irrelevant. A good example is the metric of new source lines of code (SLOC) produced per person-day on a large project. The value of this metric continues to hover around 10, giving an impression of no progress. However if one looks at Chart 21, reflecting several decades of experience at Bell Laboratories summarized in [Bernstein, 1997], the number of executing lines of machine code generated by a line of source code has increased by roughly an order of magnitude every 20 years.
Even stronger progress indicators can be generated by combining productivity per IT platform by the number of platforms in use. Effective examples in the computer hardware and communications field are plots showing high exponential growth in number of transistors in use; in petaflops of computing power available; or in numbers of Internet packets handle per year.

A software counterpart using similar counting rules is lines of code in service (LOCS), obtained by multiplying executable lines of machine code per computing platform by the number of platforms in use. Chart 22 shows the trend in LOCS over several decades of experience in the U.S. Department of Defense [Boehm, 1999]. Here, LOCS have increased by an order of magnitude roughly every 7 years, and cost per LOCS is currently decreasing at about the same rate.

Metrics such as LOCS, transistors, petaflops, and packets at least pass a market test for value, since they are items that people and organizations have paid market prices to obtain. Beyond this, however, one would prefer metrics which more closely reflect value added to people and organizations. As discussed in Victor Basili’s “Modeling and Analysis” presentation below, such metrics are emerging for software, but there are challenges in applying them across different application areas, and across stakeholders having different values and priorities.

This presents another challenge in “weaving the new information fabric:” automated collection and analysis of usage experience data from Larman and computer sources. As discussed in Richard Taylor’s “Connectivity and Information Access” presentation, capabilities will be available to sample and analyze billions of concurrent transactions. This capability can be harnessed not only for better personal, business, and government decisions, but also for analyzing and improving the various dimensions of effectiveness of the future software and IT systems we produce.
References


