

INPUT FOR NITRD FIVE-YEAR PLAN AUGUST 2008

Cyber-Physical Systems

This paper could apply to any area of software practice, but its suggestions are especially important for Cyber-Physical Systems (CPS). The area deserves the effort. By their embedded nature, CPS are low profile, and the area has been under-served by CS/SE research. However, CPS are critical to many US industries, they are knitted into the modern lifestyle, and defective cyber-physical systems can be deadly. This paper focuses on exploiting the potential of interagency projects with the cooperation of industry partners in ways that could ameliorate long-standing problems that are especially troublesome to CPS.

This paper suggests strategies to address two complaints:

1. Researchers argue that their ideas improve the software development process, but their research doesn't provide convincing evidence. They ask us to "bet the company" on new ideas supported by anecdotes and arguments. If they have experimental evidence, it is often based on experience with a few students working on small projects. Almost all embedded software developers are deeply conservative about the way they build software. They need convincing reassurance before they will adopt new technologies.
2. Either researchers don't attack the problems that bother practitioners, they have addressed those problems and the results are not implemented in commercially available software and well-documented methodologies, or wonderful things have been offered to us in attractive packages and we've ignored them. The latter is unfortunately likely. (See the first complaint.)

Validate Research Results Using Real Software Development

Any large software project could be an opportunity to test and compare development tools and methodologies. Some large software projects have been used to test new ideas, but I don't know of a case where a real software project was used to compare competing ideas (except, maybe, the control software for the space shuttle). The data generated by a comparison is more useful than the data from a test. To take an example that would be of special interest to me, a test can show that the RTSJ is useful for a large real-time software project, but a comparison would show its strengths and weaknesses relative to alternative technologies.

A software development project can be used to generate comparative data by duplicating the appropriate part of the project in at least two separate and equivalent sets of groups using different tools/methodologies/whatever. To be statistically valid, there should be several groups in each set. ¹ This approach can be used to get experimental data on many questions that trouble software engineers, but it will be expensive. A single instance of a "real" software project may cost

¹ Maybe statistical trickery can let one experiment answer several questions, giving better value for these experiments.

millions of dollars. Replicating it n times with suitable experimental measurement and isolation could cost more than n times the cost of a single instance.

Interagency cooperation, and cooperation with industry, may make suitable projects available for this kind of research. DoD, automotive companies, NASA, aerospace companies, DOE, and FAA have large, demanding software projects that would be excellent candidates for experimentation. Other problems include: finding worthwhile questions whose answers are measurable (statistics only helps when there's something meaningful to measure), and convincing the owners of the projects to let them be used this way.

This type of experiment seems likely to be most useful in two scenarios: evaluating broad research directions early in the R&D process, and demonstrating the usefulness of ideas that seem powerful to the research community but are not immediately accepted by practitioners.

Build Research and Development Communities

From a pragmatic viewpoint, research doesn't matter unless it improves real software products, or makes software production more efficient. From this viewpoint some aspects of operating system software, compiler software, software project management, data base technology, and networking get attention. Other important areas are mainly neglected.

I spend much of my life testing and debugging software, but over the past 25 years few breakthroughs have appeared in the testing/debugging software that I use. I wish those products were advancing faster. If CS/SE R&D were proportioned according to my workload, about half its effort would be directed at improving testing and debugging technology.

It might help focus researchers' efforts if CS/SE research groups were tightly coupled to users. Tying the groups together would make it easy for the researchers to understand the problems facing developers, and create a tight feedback loop that might engender useful new research and uncover ideas that have been composting in research libraries. A "real" software project might resist being asked to use tools with less-than-professional polish. That problem can be lessened by including a professional software company in the community.

I envision a systems group from CMU dedicated to providing improved systems software for the Mars lander team at JPL, with the JPL team having a matching commitment to depend on systems software from the CMU team, and a software product team from IBM putting a professional polish on the CMU software and providing first rate documentation and support. It might be best if the three groups were located together.

The teams should collaborate through several generations of the product.

Not all important work is done by large groups. It would be hard, but useful, to build communities to focus researchers' attention on tools and methodologies for development teams with fewer than six members.