Rapid change in information technology brings with it a frequent need to undo the effects of previous culture change efforts. This process, while often challenging and frustrating, offers numerous rewards for success. Organizations can change from slow, reactive, adversarial, separated software and systems engineering processes to unified, concurrent processes. These processes better suit rapid development of dynamically changing software-intensive systems involving COTS, agent, Web, multimedia, and Internet technology.

Culture changes are never easy, however. For example, in the mid-1970s, I participated in a group at TRW that created a corporate software engineering culture around the sequential requirements-driven waterfall model. The effort included corporate policies and standards, training courses administered by managers, and even stringent test on software policies.

We did not realize how effective this culture change was until we tried to undo parts of it a few years later. By the early 1980s, we and other companies realized that the waterfall model was ineffective for developing user-interactive systems, in which prototypes rather than exhaustive processing specifications proved more effective in determining the most appropriate product characteristics. The “A Large Sequential-Engineering Near-Disaster” sidebar provides a good example of the mismatch we found between the sequential, deductive waterfall approach and the need for emergent, synthetic approaches for user-intensive systems.

When we attempted to introduce prototyping into user-intensive projects at TRW, software engineers, resistant to this change, pounded the table and shouted, “You can’t do that! Prototyping is coding before you’ve passed your Critical Design Review, which corporate policies absolutely forbid!” It took years to undo the many ways in which the waterfall model had percolated into various aspects of our corporate culture such as marketing, proposal preparation, cost estimating, contracting, accounting, and management.

The recent initiatives to integrate the CMU-SEI Software Capability Maturity Model that integrated the software engineering, systems engineering, and acquisition CM M s. The second effort is the SEI and Department of Defense-led CMM Integration (CMIM) project, which currently integrates the software engineering, systems engineering, and integrated product and process development CMMs, and provides a framework for integrating other CMMs.

ROOTING OUT THE WATERFALL LEGACY

Even the current software CMM (version 1.1), although a great improvement over previous undisciplined practices, reinforces the sequential waterfall process. These legacy reinforcements frustrate and delay transition toward effective means of developing user-interactive systems—early prototyping, concurrent engineering, risk management, integrated product teams, and other steps toward unifying software and systems engineer-
requirements. The quote implies that software people were not needed or wanted for system engineering activities.

I observed the social consequences of this approach in several aerospace system-architecture-definition meetings ("Integrating Software Engineering and System Engineering," Journal of IN CO SE, Jan. 1994, pp. 61-67). While the hardware and systems engineers sat around the table discussing their previous system architectures, the software engineers sat on the side, waiting for someone to give them a precise specification they could turn into code.

This separation of concerns has largely kept software engineering methods focused on abstract logical exercises, while in practice, the software drives system considerations such as performance and cost. For example, in a recent survey of 16 books on O O design, only six had the word “performance” in their index, and only two had the word “cost.” Using such abstract logical exercises carries with it a major risk of software-system disasters or near-disasters such as the one described in the sidebar.

INTEGRATED CMM IMPROVEMENTS

The current draft Capability Maturity Model—Integrated Systems/Software Engineering (CMM I—SE/SW) and its recent draft extension to accommodate Integrated Process and Product Development (IPPD) can be found at http://www.sei.cmu.edu/cmm/cmmi/. The entire package introduces a new paradigm for concurrent system and software engineering.

In it, three of the new IPPD process areas—Integrated Team, Shared Vision, and Collaborative Leadership—establish the stakeholder collaboration preconditions for concurrent system and software engineering. The particular concurrent engineering practices are established via combined SW, SE, and IPPD process areas of Customer and Product Requirements, Technical Solution, Project Planning, Supplier Agreement Management, Risk Management, and Decision Analysis and Resolution.

At the next level of detail, the Customer and Product Requirements process area no longer impels software engineers into a reactive role of allocated software

A Large Sequential-Engineering Near-Disaster

In the early 1980s, a large government organization contracted with TRW to develop an ambitious information query and analysis system. The system would provide more than 1,000 users, spread across a large building complex, with powerful query and analysis capabilities for a large and dynamic database.

TRW and the customer specified the system using a classic sequential-engineering waterfall development model. Based largely on user need surveys and high-level performance analysis, they required a system response time of less than one second.

Two thousand pages of requirements later, the software architects found that subsecond performance could only be provided via a highly customized design that attempted to anticipate query patterns and cache copies of data so that each user’s likely data would be within one second’s reach. The resulting hardware architecture had more than 25 super-midicomputers busy caching data according to algorithms whose actual performance defied easy analysis. The scope and complexity of the hardware-software architecture brought the estimated cost of the system to nearly $100 million, driven primarily by the requirement for a one-second response time.

Faced with this unattractive prospect, the customer and developer decided to develop a prototype of the system’s user interface and representative capabilities to test. The results showed that a four-second response time would satisfy users 90 percent of the time. A four-second response time dropped development costs closer to $30 million.

Figure A shows a performance comparison of the initial (I) and final (F) architectures.

Besides illustrating the dangers of separating software engineering from system requirements specification, the comparison shown in Figure A provides several additional points about the relationship between information system requirements and architectures:

• Nonfunctional (often called property, quality, or level-of-service) requirements influence preferred architectures.
• The preferred architecture is often a discontinuous function of the nonfunctional requirements. Somewhere between response times of 1 and 4 seconds the preferred architecture changes radically from I to F.
• Using the size of the requirements specification to estimate cost can be risky. In this case, changing one character (a “1” to a “4”) in a specification reduced the cost by a third.

Figure A. Performance comparison of initial and final architectures.
requirements. Instead, they’re involved in eliciting stakeholder needs, expectations, and constraints; transforming them into customer requirements; obtaining developer and acquirer agreements; developing operational concepts and scenarios; establishing product requirements, architecture, and interfaces; validating requirements; reducing cost and risk; and analyzing adequacy.

**UNIFIED PROJECT APPROACHES**

The CMMI thus provides software engineers with a seat at the center table. It also identifies the additional activities necessary for successful concurrent engineering of systems and software. However, it leaves open the definition of project workflows and methods to be used in performing the activities. Developers can now choose from several approaches built on earlier partial initiatives to integrate system and software engineering activities.

The Software Productivity Consortium has two processes that draw on the spiral model: Both the Evolutionary Spiral Process and Integrated Systems and Software Engineering Process utilize the SPC’s past experience in defining a systems engineering CMM and providing assessment and training for the software CMM.

The Rational Unified Process (RUP) (Walker Royce, Software Project Management: A Unified Framework, Addison Wesley, 1998) and USC’s Model-Based (System) Architecting and Software Engineering (M BASE) (Barry Boehm, Dan Port, “Escaping the Software Tar Pit: Model Clashes and How to Avoid Them,” ACM Software Engineering Notes, January 1999, pp. 36-48; see also http://sunset.usc.edu/M BASE) share a common set of integrated systems, software engineering phases (inception and elaboration), and stakeholder commitment milestones (life cycle objectives and life cycle architecture). However, whereas RUP is more strongly coupled to and supported by Rational’s systems and SE toolset, M BASE receives support from the WinWin stakeholder requirements negotiation system and an electronic process guide. M BASE is also mapped to the CMM I process areas, and emphasizes economic business case analysis.

The DMR Group’s Benefits Realization Approach (DMR-BRA) (John Thorp, The Information Paradox, McGraw Hill, 1998) goes even further. It not only supports the initial business case analysis, linking software and systems engineering with business management, but also provides a closed-loop feedback process that monitors estimated benefits in the business case. The monitoring process is based on a results chain that links project initiatives, resulting contributions, desired outcomes, and assumptions upon which successful outcomes depend. The DMR-BRA phase gates resemble the RUP and M BASE stakeholder commitment milestones, making the approaches feasible to use together.

The CMM I and the emerging project methods I’ve described demonstrate the opportunities for process improvement gains open to organizations. The organization that changes from separated software and system engineering processes to a more unified approach will find itself far more suited to developing dynamically changing, software-intensive systems. Culture change is never easy, but the alternative is even less palatable.

Barry Boehm is director of the USC Center for Software Engineering. Contact him at boehm@sunset.usc.edu.