Some Future Trends and Implications for Systems and Software Engineering Processes

Barry Boehm*

Center for Software Engineering, University of Southern California, Los Angeles, CA 90089-0781

ABSTRACT

In response to the increasing criticality of software within systems and the increasing demands being put onto 21st century systems, systems and software engineering processes will evolve significantly over the next two decades. This paper identifies eight relatively surprise-free trends—the increasing interaction of software engineering and systems engineering; increased emphasis on users and end value; increased emphasis on systems and software dependability; increasingly rapid change; increasing global connectivity and need for systems to interoperate; increasingly complex systems of systems; increasing needs for COTS, reuse, and legacy systems and software integration; and computational plenty. It also identifies two "wild card" trends: increasing software autonomy and combinations of biology and computing. It then discusses the likely influences of these trends on systems and software engineering processes between now and 2025, and presents an emerging scalable spiral process model for coping with the resulting challenges and opportunities of developing 21st century software-intensive systems and systems of systems. © 2006 Wiley Periodicals, Inc. Syst Eng 9: 1–19, 2006

Key words: future trends; systems engineering; software engineering; systems architecting; human systems integration; systems acquisition; systems of systems; spiral model; value-based processes

1. INTRODUCTION

Between now and 2025, the ability of organizations and their products, systems, and services to compete, adapt, and survive will depend increasingly on software. As is being seen in current products (automobiles, aircraft, radios) and services (financial, communications, de-

fense), software provides both competitive differentiation and rapid adaptability to competitive change. It facilitates rapid tailoring of products and services to different market sectors, and rapid and flexible supply chain management. The resulting software-intensive systems face ever-increasing demands to provide safe, secure, and reliable systems; to provide competitive discriminators in the marketplace; to support the coordination of multicultural global enterprises; to enable rapid adaptation to change; and to help people cope with complex masses of data and information. These demands will cause major differences in the processes currently used to define, design, develop, deploy, and evolve a diverse variety of software-intensive systems.

This paper elaborates on the nature of these increasing demands on software-intensive systems and their process implications. Section 2 will identify eight relatively surprise-free trends and two less-predictable "wild card" trends, and discuss their likely influence on systems and software engineering processes between now and 2025. Section 3 presents the emerging scalable spiral process model for coping with the trends discussed in Section 2. It involves two coordinated teams concurrently performing stabilized plan-driven development and verification/validation of the current system increment, while a third agile rebaselining team concurrently evolves the changing requirements, product plans, and process plans for the next increment. Section 4 identifies and discusses some 20th century processes and institutions that need to be significantly rethought and reworked to cope with the 21st century systems, particularly acquisition practices and human relations. Section 5 presents the resulting conclusions.

2. SOFTWARE-INTENSIVE SYSTEMS (SIS) TRENDS AND THEIR INFLUENCE ON SYSTEMS AND SOFTWARE ENGINEERING PROCESSES

The eight relatively surprise-free trends are:

1. The increasing integration of software engineering and systems engineering
2. An increased emphasis on users and end value
3. Increasing SIS criticality and need for dependability
4. Increasingly rapid change
5. Increasing SIS globalization and need for interoperability
6. Increasingly complex systems of systems
7. Increasing needs for COTS, reuse, and legacy SIS integration
8. Computational plenty.

The two "wild card" trends are:

9. Increasing software autonomy
10. Combinations of biology and computing.

2.1. Increasing Integration of Software Engineering and Systems Engineering

Several trends have caused systems engineering and software engineering to initially evolve as largely sequential and independent processes. First, systems engineering began as a discipline for determining how best to configure various hardware components into physical systems such as ships, railroads, or defense systems. Once the systems were configured and their component functional and interface requirements were precisely specified, sequential external or internal contracts could be defined for producing the components. When software components began to appear in such systems, the natural thing to do was to treat them sequentially and independently as Computer Software Configuration Items.

Second, the early history of software engineering was heavily influenced by a highly formal and mathematical approach to specifying software components, and a reductionist approach to deriving computer software programs that correctly implemented the formal specifications. A "separation of concerns" was practiced, in which the responsibility of producing formalizable software requirements was left to others, most often hardware-oriented systems engineers. Some example quotes illustrating this approach are:

- "The notion of 'user' cannot be precisely defined, and therefore has no place in computer science or software engineering," E. W. Dijkstra, panel remarks, ICSE 4, 1979 [Dijkstra, 1979].
- "Analysis and allocation of the system requirements is not the responsibility of the software engineering group but is a prerequisite for their work," CMU-SEI Software Capability Maturity Model, version 1.1, 1993 [Paulk et al., 1994].

As a result, a generation of software engineering education and process improvement goals were focused on reductionist software development practices that assumed that other (mostly nonsoftware people) would finish appropriate predetermined requirements for the software.

Third, the business practices of contracting for components were well worked out. Particularly in the government sector, acquisition regulations, specifications, and standards were in place and have been traditionally difficult to change. The path of least resistance was to follow a "purchasing agent" metaphor and sequentially specify requirements, establish contracts, formulate and implement solutions, and use the requirements to acceptance-test the solutions [USD, 1985, 1988]. When

requirements and solutions were not well understood or changing rapidly, knowledgeable systems and software engineers and organizations could reinterpret the standards to operate more flexibly, concurrently and pragmatically and to produce satisfactory systems [Checkland, 1999; Royce, 1998]. But all too frequently, the sequential path of least resistance was followed, leading to the delivery of obsolete or poorly performing systems.

As the pace of change increased and systems became more user-intensive and software-intensive, serious strains were put on the sequential approach. First, it was increasingly appreciated that the requirements for user-intensive systems were generally not prespecifiable in advance, but emergent with use. This undermined the fundamental assumption of sequential specification and implementation.

Second, having people without software experience determine the software specifications often made the software much harder to produce, putting software even more prominently on the system development's critical path. Systems engineers without software experience would minimize computer speed and storage costs and capacities, which causes software costs to escalate rapidly [Boehm, 1981]. They would choose best-of-breed system components whose software was incompatible and time-consuming to integrate. They would assume that adding more resources would speed up turnaround time or software delivery schedules, not being aware of slowdown phenomena such as multiprocessor overhead [Boehm, 1981] or Brooks' Law (adding more people to a late software project will make it later) [Brooks, 1995].

Third, software people were recognizing that their sequential, reductionist processes were not conducive to producing user-satisfactory software, and were developing alternative software engineering processes (evolutionary, spiral, agile) involving more and more systems engineering activities. Concurrently, systems engineering people were coming to similar conclusions about their sequential, reductionist processes, and developing alternative "soft systems engineering" processes [e.g., Checkland, 1999], emphasizing the continuous learning aspects of developing successful user-intensive systems. Similarly, the project management field is undergoing questioning about its underlying specification-planning-execution-control theory being obsolete and needing more emphasis on adaptation and value generation [Koskela and Howell, 2002].

2.1.1. Systems and Software Engineering Process Implications

Many commercial organizations have developed more flexible and concurrent development processes [Womack, Jones, and Roos, 1990]. Also, recent process

guidelines and standards such as the Integrated Capability Maturity Model (CMMI) [Chrissis, Konrad, and Shrum, 2003], ISO/IEC 12207 for software engineering [ISO, 1995], and ISO/IEC 15288 for systems engineering [ISO, 2002] emphasize the need to integrate systems and software engineering processes, along with hardware engineering processes and human engineering processes. They emphasize such practices as concurrent engineering of requirements and solutions, integrated product and process development, and risk-driven vs. document-driven processes. New process milestones enable effective synchronization and stabilization of concurrent processes [Boehm, 1996; Kroll and Kruchten, 2003].

However, contractual acquisition processes still lag behind technical processes. Many organizations and projects develop concurrent and adaptive development processes, only to find them frustrated by progress payments and award fees emphasizing compliance with sequential document-driven deliverables. As we will discuss next, though, the increasing emphasis on end-user value and the need for rapidly evolving systems will continue to drive the full integration and concurrency of software engineering and systems engineering processes. And in Section 3 we will provide a process framework for integrating systems and software engineering of 21st century systems, and for improving contractual acquisition processes.

2.2. User/Value Emphasis Trends and Process Implications

A recent Computerworld panel on "The Future of Information Technology (IT)" indicated that usability and total ownership cost-benefits, including user inefficiency and ineffectiveness costs, are becoming IT user organizations' top priorities [Anthes, 2005]. A representative quote from panelist W. Brian Arthur was "Computers are working about as fast as we need. The bottleneck is making it all usable." A recurring user-organization desire is to have technology that adapts to people rather than vice versa. This is increasingly reflected in users' product selection activities, with evaluation criteria increasingly emphasizing product usability and value added vs. a previous heavy emphasis on product features and purchase costs. Such trends ultimately will affect producers' product and process priorities, marketing strategies, and competitive survival.

Some technology trends strongly affecting usability and cost-effectiveness are increasingly powerful enterprise support packages, data access and mining tools, and Personal Digital Assistant (PDA) capabilities. Such products have tremendous potential for user value, but

determining how they will be best configured will involve a lot of product experimentation, shakeout, and emergence of superior combinations of system capabilities.

2.2.1. Systems and Software Engineering Process Implications

In terms of future systems and software process implications, the fact that the capability requirements for these products are emergent rather than prespecifiable has become the primary challenge. Not only do the users exhibit the IKIWISI (I'll know it when I see it) syndrome, but their priorities change with time. These changes often follow a Maslow need hierarchy, in which unsatisfied lower-level needs are top priority, but become lower priorities once the needs are satisfied [Maslow, 1954]. Thus, users will initially be motivated by survival in terms of capabilities to process new workloads, followed by security once the workload-processing needs are satisfied, followed by self-actualization in terms of capabilities for analyzing the workload content for self-improvement and market trend insights once the security needs are satisfied. Chapter 1 of the recent *Handbook of Human Systems Integration* [Booher, 2003] summarizes the increased emphasis on human factors integration into systems engineering, and its state of progress in several large government organizations.

It is clear that requirements emergence is incompatible with past process practices such as requirements-driven sequential waterfall process models and formal programming calculi, and with process maturity models emphasizing repeatability and optimization [Paulk et al., 1994]. In their place, more adaptive [Highsmith, 2000] and risk-driven [Boehm, 1988] models are needed. More fundamentally, the theory underlying software and systems engineering process models needs to evolve from purely reductionist "modern" world views (universal, general, timeless, written) to a synthesis of these and situational "postmodern" world views (particular, local, timely, oral) as discussed in [Toulmin, 1992]. A recent theory of value-based software engineering (VBSE) and its associated software processes [Boehm and Jain, 2005] provide a starting point for addressing these challenges, and for extending them to systems engineering processes. The associated VBSE book [Biffl et al., 2005] contains further insights and emerging directions for VBSE processes.

For example, the chapter on "Stakeholder Value Proposition Elicitation and Reconciliation" in the VBSE book [Biffl et al., 2005] addresses the need to evolve from software products, methods, tools, and educated students strongly focused on individual performance to a focus on more group-oriented interdisciplinary collaboration. Negotiation of priorities for requirements involves not only participation from users and acquirers on each requirement's relative mission or business value, but also participation from systems and software engineers on each requirement's relative cost and time to develop and difficulty of implementation. As we will discuss further under Globalization in Section 2.4, collaborative activities such as Participatory Design [Ehn, 1990] will require stronger human systems and software engineering process and skill support not only across application domains but also across different cultures.

The aforementioned *Handbook of Human Systems Integration* [Booher, 2003] identifies a number of additional principles and guidelines for integrating human factors concerns into the systems engineering process. In particular, it identifies the need to elevate human factor concerns from a micro-ergonomics to a macro-ergonomics focus on organization, roles, responsibilities, and group processes of collective observation, orientation, decision-making, and coordinated action.

Some additional process implications of user- and value-oriented trends are addressed in the ICSE 2000 Software Process Roadmap [Fuggetta, 2000]. They include the need for process modeling languages to allow for incomplete, informal, and partial specification; the need for process-centered environments to be incremental and ambiguity-tolerant; and for software process metrics and empirical studies to be value-driven. Similar user- and value-oriented trends for systems engineering are addressed in the draft INCOSE Technical Vision document [INCOSE, 2005].

2.3. Systems and Software Criticality and Dependability Trends

Although people's, systems', and organizations' dependency on software is becoming increasingly critical, the current major challenge in achieving system dependability is that dependability is generally not the top priority for software producers. In the words of the 1999 PITAC Report, "The IT industry spends the bulk of its resources, both financial and human, on rapidly bringing products to market" [PITAC, 1999].

Recognition of the problem is increasing. ACM President David Patterson has called for the formation of a top-priority Security/Privacy, Usability, and Reliability (SPUR) initiative [Patterson, 2005]. Several of the Computerworld "Future of IT" panelists [Anthes, 2005] indicated increasing customer pressure for higher quality and vendor warranties, but others did not yet see significant changes happening among software product vendors.

This situation will likely continue until a major software-induced systems catastrophe similar in impact on world consciousness to the 9/11 World Trade Center catastrophe stimulates action toward establishing accountability for software dependability. Given the high and increasing software vulnerabilities of the world's current financial, transportation, communications, energy distribution, medical, and emergency services infrastructures, it is highly likely that such a software-induced catastrophe will occur between now and 2025.

2.3.1. Systems and Software Process Implications

Process strategies for highly dependable software-intensive systems and many of the techniques for addressing its challenges have been available for quite some time. A landmark 1975 conference on reliable software included papers on formal specification and verification processes; early error elimination; fault tolerance; fault tree and failure modes and effects analysis; testing theory, processes and tools; independent verification and validation; root cause analysis of empirical data; and use of automated aids for defect detection in software specifications and code [Boehm and Hoare, 1975]. Some of these were adapted from existing systems engineering practices; some were developed for software and adapted for systems engineering.

These have been used to achieve high dependability on smaller systems and some very large self-contained systems such as the AT&T telephone network [Musa, 1999]. Also, new strategies have been emerging to address the people-oriented and value-oriented challenges discussed in Section 2.1. These include the Personal and Team Software Processes [Humphrey, 1997, 2000], value/risk-based processes for achieving dependability objectives [Gerrard, 2002; Huang, 2005], and value-based systems engineering processes such as Lean Development [Womack and Jones, 1996].

The major future challenges for systems and software dependability processes are in scaling up and integrating these approaches in ways that also cope with the challenges presented by other future trends. These include the next four trends to be discussed below: rapid change and agility; globalization; complex systems of systems; and COTS/legacy integration. The remaining trends—computational plenty, autonomy, and biocomputing—will offer further attractive solution avenues, but also further challenges, as we will also discuss below.

2.4. Rapid Change Trends

The increasingly rapid pace of systems change is driven by technology trends such as Gordon Moore's Law (transistor and integrated circuit density and performance doubles roughly every 18 months), plus the continuing need for product differentiation and tornadolike processes for new technology introduction [Geoffrey Moore, 1995]. Global connectivity also accelerates the ripple effects of technology, marketplace, and technology changes. Rapid change also increases the priority of development speed vs. cost in capitalizing on market windows. A good example of systems engineering process improvement toward rapid change was Toyota's lean automotive design and production processes [Womack, Jones, and Roos, 1990]. A good example of successful software process improvement toward rapid change was Hewlett Packard's initiative to reduce product line software development times from 48 to 12 months [Grady, 1997].

When added to the trend toward emergent systems requirements, the pace of change places a high priority on systems and software engineering process agility and investments in continuous learning for both people and organizations. Such investments need to be organization-wide. Many organizations have invested in agility and continuous learning for their technical people, but have left their administrative and contract people to propagate the THWADI (That's how we've always done it) syndrome. This usually leads to unpleasant surprises when their agile technical processes run afoul of slow, outdated, and inflexible procurement or approval procedures.

Completely eliminating THWADI is not a good idea either, as it includes an organization's corporate memory and best practices. The big challenge is to determine which legacy processes and principles to keep, modify, or eliminate.

2.4.1. Systems and Software Engineering Process Implications

In the software-intensive systems field, agile methods such as Extreme Programming [Beck, 1999], Adaptive Software Development [Highsmith, 2000], and Crystal [Cockburn, 2002] have shown considerable success in adapting to rapid change on small-to-medium size projects. One of their primary keys to success is to staff and develop teams (including dedicated and collocated customers) that can operate on shared tacit knowledge vs. explicit documented knowledge. Another is to start right away to develop short, stabilized increments of system capability, and to use the next increment to quickly remove problems in the current increment. Others are to develop test cases first as executable versions of requirements, to quickly refactor software to adapt to unforeseen change, and to run the test cases to ensure that the refactoring preserved the system capability. Our analysis of agile methods experiences

to date in Balancing Agility and Discipline [Boehm and Turner, 2004] indicated that the techniques that enable agile methods to work well on small-to-medium size projects can become sources of difficulty when applied to larger, more mission-critical projects. Figure 1 identifies five key dimensions that we found to characterize the "home grounds" in which agile methods and plan-driven methods work best:

- Project size, in terms of number of project personnel
- Mission criticality, in terms of the effect of a system failure
- Personnel skills, in terms of a 3-level capability scale
- Project Dynamism, in terms of percent of requirements change per month
- Organizational culture, in terms of seeking to thrive on order vs. chaos.

Projects rated at the two levels near the center for each dimension are within the agile home ground but are not well suited for plan-driven methods. The opposite is true for projects rated at the two levels at the outside. The profile in Figure 1 is of the ThoughtWorks Lease Management Project, which did quite well initially as an agile project but began to experience difficulties as its size reached 50 people, 500,000 source lines of code, and more mission critical performance [Elssamadisy and Schalliol, 2002]. In order to continue to succeed, the project developed a hybrid mix of agile and plan-driven methods, including high-level architecture documentation and more explicit increment plans.

A major challenge for the future is to determine ways of balancing agile and plan-driven methods for such mixed-profile projects. An overall framework for achieving such a balance is provided in [Boehm and Turner, 2004].

As shown in Figure 2, it involves a risk-driven spiral approach to determine whether a primarily agile or primarily plan-driven development approach is best, or if both are needed. If the latter, an effective architectural approach involves identifying the parts of the application most needing agility, and encapsulating them within a plan-driven framework, using the Parnas technique [Parnas, 1979] of encapsulating sources of change within architected modules. Frequent examples of encapsulated agile system parts are the parts dealing with unpredictable change in user interfaces or external system interfaces. A critical factor is the ability to pass a Life Cycle Architecture (LCA) milestone review [Boehm, 1996] before proceeding from concept exploration to development.

For systems already in operation, a rapid pace of change requires a pro-active, risk-driven monitoring and experimentation activity to identify and prepare for major changes. An excellent systems engineering example is the spiral process used to evolve the ARPAnet and Internet (even before the spiral model was defined and published). Using the approach shown in Figure 3, the ARPAnet/Internet principals were able to experiment, pilot, evaluate, and redefine the Internet standards to evolve from wire-based communication through packet radio, satellite, and optical communication while providing continuity of service [Druffel et al., 1994]. A good software engineering example of tech-
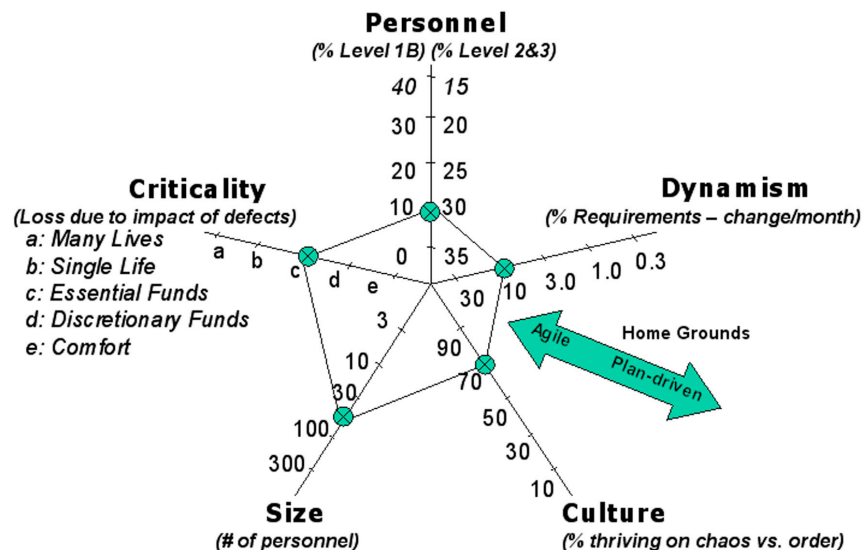

Figure 1. ThoughtWorks Lease Management Project: Agile and plan-driven home grounds and mixed-project example.
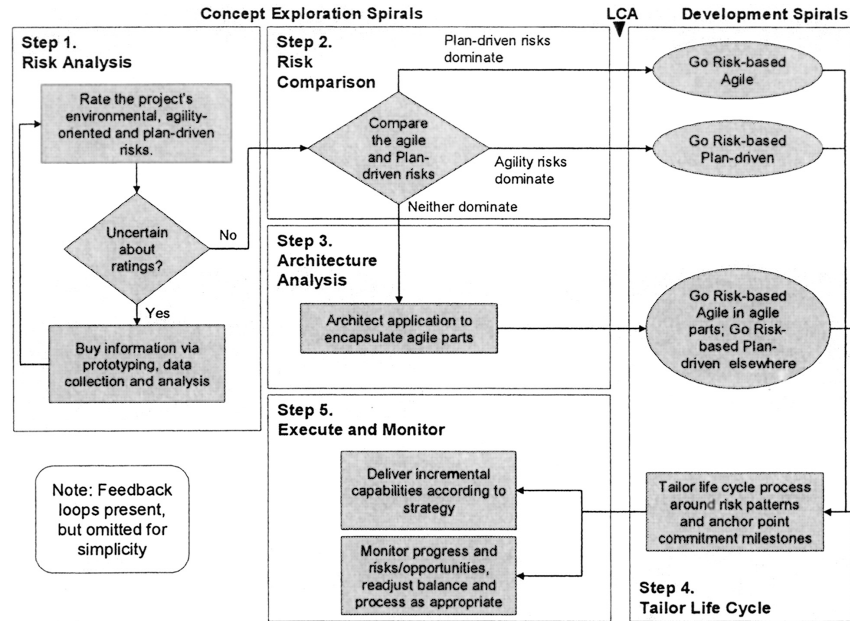
Figure 2. Risk-driven spiral approach to balancing agility and discipline.

nology monitoring, empirical experimentation, and process improvement was the NASA Goddard/University of Maryland/CSC Software Engineering Laboratory [Basili et al., 1995].

Rapid change prompts an additional significant change in systems and software engineering education. This is to help students not only learn concepts, processes, and techniques, but also to learn how to learn. In some of our systems and software engineering courses, we have addressed this challenge by adding assignments for students to obtain information from as many sources as possible to evaluate the maturity of an emerging technology and recommend a strategy that an organization could use with respect to adoption of the
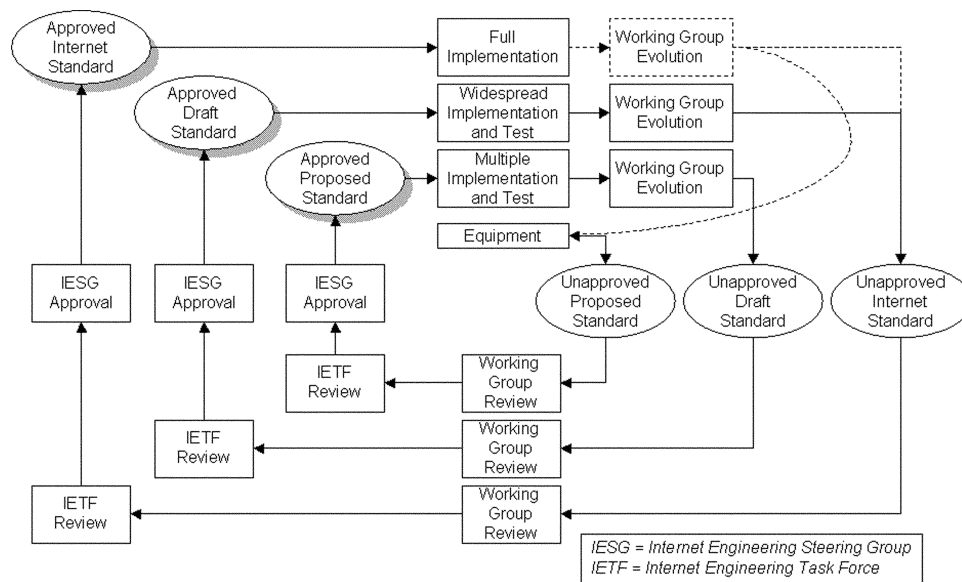
Figure 3. The Internet Spiral Process [Druffel et al., 1994].

technology. We are also exploring other strategies for helping students learn how to learn, such as the use of game technology.

2.5. Globalization and Interoperability Trends

The global connectivity provided by the Internet provides major economies of scale and network economies [Arthur, 1996] that drive both an organization's product and process strategies. Location-independent distribution and mobility services create both rich new bases for synergetic collaboration and challenges in synchronizing activities. Differential salaries provide opportunities for cost savings through global outsourcing, although lack of careful preparation can easily turn the savings into overruns. The ability to develop across multiple time zones creates the prospect of very rapid development via three-shift operations, although again there are significant challenges in management visibility and control, communication semantics, and building shared values and trust.

On balance, though, the Computerworld "Future of IT" panelists felt that global collaboration would be commonplace in the future. The primary driver would evolve away from cost differentials as they decrease, and evolve toward the complementarity of skills in such areas as culture-matching and localization [Anthes, 2005]. Some key culture-matching dimensions are provided in Hofstede [1997]: power distance, individualism/collectivism, masculinity/femininity, uncertainty avoidance, and long-term time orientation. These often explain low software product and process adoption rates across cultures. One example is the low adoption rate (17 out of 380 experimental users) of the more individual/masculine/short-term U.S. culture's Software CMM by organizations in the more collective/feminine/long-term Thai culture [Phongpaibul and Boehm, 2005]. Another example was a much higher Chinese acceptance level of a workstation desktop organized around people, relations, and knowledge as compared to Western desktop organized around tools, folders, and documents [proprietary communication].

As with railroads and telecommunications, a standards-based infrastructure is essential for effective global collaboration. The Computerworld panelists envision that standards-based infrastructure will become increasingly commoditized and move further up the protocol stack toward applications.

2.5.1. Systems and Software Engineering Process Implications

A lot of work needs to be done to establish robust success patterns for global collaborative processes. Key challenges as discussed above include cross-cultural bridging; establishment of common shared vision and trust; contracting mechanisms and incentives; handovers and change synchronization in multi-timezone development; and culture-sensitive collaboration-oriented groupware. Most software packages are oriented around individual use; just determining how best to support groups will take a good deal of research and experimentation.

One collaboration process whose future applications niche is becoming better understood is open-source software development. Security experts tend to be skeptical about the ability to assure the secure performance of a product developed by volunteers with open access to the source code. Feature prioritization in open source is basically done by performers; this is generally viable for infrastructure software, but less so for competitive corporate applications systems and software. Proliferation of versions can be a problem with volunteer developers. But most experts, including the Computerworld futures panel, see the current success of open source development for infrastructure products such as Linux, Apache, and Firefox as sustainable into the future.

2.6. Complex Systems of Systems Trends

Traditionally (and even recently for some forms of agile methods), systems and software development processes were recipes for standalone "stovepipe" systems with high risks of inadequate interoperability with other stovepipe systems. Experience has shown that such collections of stovepipe systems cause unacceptable delays in service, uncoordinated and conflicting plans, ineffective or dangerous decisions, and inability to cope with rapid change.

During the 1990s and early 2000s, standards such as ISO/IEC 12207 [ISO, 1995] and ISO/IEC 15288 [ISO, 2002] began to emerge that situated systems and software project processes within an enterprise framework. Concurrently, enterprise architectures such as the IBM Zachman Framework [Zachman, 1987], RM-ODP [Putman, 2001], and the U.S. Federal Enterprise Framework [FCIO, 2001], have been developing and evolving, along with a number of commercial Enterprise Resource Planning (ERP) packages.

These frameworks and support packages are making it possible for organizations to reinvent themselves around transformational, network-centric systems of systems. As discussed in Harned and Lundquist [2003], these are necessarily software-intensive systems of systems (SISOS), and have tremendous opportunities for success and equally tremendous risks of failure. Examples of successes have been Federal Express, Wal-Mart, and the U.S. Command, Control, Intelligence, Surveil-

lance, and Reconnaissance (C2ISR) system in Iraq; examples of failures have been the Confirm travel reservation system, K-Mart, and the U.S. Advanced Automation System for air traffic control. ERP packages have been the source of many successes and many failures, implying the need for considerable risk/opportunity assessment before committing to an ERP-based solution.

2.6.1. Systems and Software Engineering Process Implications

Our work in supporting SISOS development programs has shown that the use of a risk-driven spiral process with early attention to SISOS risks and systems architecting methods [Rechtin, 1991] can avoid many of the SISOS development pitfalls [Boehm et al., 2004]. A prioritized list of the top ten SISOS risks we have encountered includes several of the trends we have been discussing: (1) acquisition management and staffing; (2) requirements/architecture feasibility; (3) achievable software schedules; (4) supplier integration; (5) adaptation to rapid change; (6) systems and software quality factor achievability; (7) product integration and electronic upgrade; (8) software COTS and reuse feasibility; (9) external interoperability; and (10) technology readiness.

In applying risk management to this set of risks, the outlines of a hybrid plan-driven/agile process architecture for developing SISOS are emerging. In order to keep SISOS developments from becoming destabilized from large amounts of change traffic, it is important to organize development into plan-driven increments in which the suppliers develop to interface specs that are kept stable by deferring changes, so that the systems can plug and play at the end of the increment (nobody has yet figured out how to do daily builds for these kinds of systems). But for the next increment to hit the ground running, an extremely agile team needs to be concurrently doing continuous market, competition, and technology watch, change impact analysis, COTS refresh, and renegotiation of the next increment's prioritized content and the interfaces between the suppliers' next-increment interface specs. This requires new approaches not just to process management, but also to staffing and contracting. Section 3 will elaborate on this emerging process architecture and its challenges.

2.7. COTS, Reuse, and Legacy Integration Challenges

A recent ACM Communications editorial stated, "In the end—and at the beginning—it's all about programming" [Crawford, 2001]. Future trends are making this decreasingly true. Although infrastructure software developers will continue to spend most of their time programming, most application software developers are spending more and more of their time assessing, tailoring, and integrating commercial off-the-shelf (COTS) products. COTS hardware products are also becoming more pervasive, although they are generally easier to assess and integrate.

Figure 4 illustrates these trends for a longitudinal sample of small e-services applications going from 28% COTS-based in 1996–1997 to 70% COTS-based in 2001–2002, plus a corroborative industry-wide 54% figure for COTS-based applications (CBAs) in the 2000 Standish Group survey [Standish Group, 2001; Yang et al., 2005]. COTS software products are particularly challenging to integrate. They are opaque and hard to debug. They are often incompatible with each other due to the need for competitive differentiation. They are uncontrollably evolving, averaging about up to 10 months between new releases, and generally unsupported by their vendors after three subsequent releases. These latter statistics are a caution to organizations outsourcing applications with long gestation periods. In one case, we observed an outsourced application with 120 COTS products, 46% of which were delivered in a vendor-unsupported state [Yang et al., 2005].

Open source software, or an organization's reused or legacy software, is less opaque and less likely to go unsupported. But such software can also have problems with interoperability and continuing evolution. In addition, it often places constraints on a new application's incremental development, as the existing software needs to be decomposable to fit the new increments' content and interfaces. Across the maintenance life cycle, synchronized refresh of a large number of continually evolving COTS, open source, reused, and legacy software and hardware becomes a major additional challenge.

In terms of the trends discussed above, COTS, open source, reused, and legacy software and hardware will
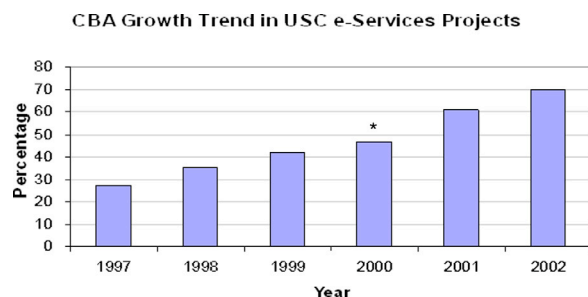

Figure 4. CBA growth in USC e-service projects [Yang et al., 2005] Standish Group, Extreme chaos [Standish Group, 2001].

often have shortfalls in usability, dependability, interoperability, and localizability to different countries and cultures. As discussed above, increasing customer pressures for COTS usability, dependability, and interoperability, along with enterprise architecture initiatives, will reduce these shortfalls to some extent.

2.7.1. Systems and Software Engineering Process Implications

COTS economics generally makes sequential waterfall processes (in which the prespecified system requirements determine the capabilities) incompatible with COTS-based solutions (in which the COTS capabilities largely determine the requirements; a desired capability is not a requirement if you can't afford the non-COTS solution that would provide it). Some initial software COTS-based applications (CBA) development processes are emerging. Some are based on composable process elements covering the major sources of CBA effort (assessment, tailoring, and glue code integration) [Yang et al., 2005; Yang, Boehm, and Port, 2005]. Others are oriented around the major functions involved in software CBAs, such as the SEI EPIC process [Albert and Brownsword, 2002].

However, there are still major challenges for the future, such as processes for synchronized multi-COTS refresh across the lifecycle; processes for enterprise-level and systems-of-systems COTS, open source, reuse, and legacy evolution; integrated hardware and software COTS-based system processes; and processes and techniques to compensate for shortfalls in multi-COTS usability, dependability, and interoperability. Such COTS shortfalls are another reason why emergent spiral or evolutionary processes are better matches to most future project situations than attempts to prespecify waterfall and V-model system and software development plans. For example, consider the following common development sequence:

1. Pick the best set of COTS products supporting the system objectives;
2. Compensate for the selected COTS shortfalls with respect to the system objectives.

There is no way to elaborate task 2 (or other tasks depending on it) until task 1 is done, as different COTS selections will produce different shortfalls needing compensation.

2.8. Computational Plenty Trends

Assuming that Moore's Law holds, another 20 years of doubling computing element performance every 18 months will lead to a performance improvement factor of $2^{20/1.5} = 2^{13.33} = 10,000$ by 2025. Similar factors will apply to the size and power consumption of the competing elements.

This computational plenty will spawn new types of platforms [smart dust, smart paint, smart materials, nanotechnology, micro electrical-mechanical systems (MEMS)], and new types of applications (sensor networks, conformable or adaptive materials, human prosthetics). These will present process-related challenges for specifying their configurations and behavior; generating the resulting applications; verifying and validating their capabilities, performance, and dependability; and integrating them into even more complex systems of systems.

2.8.1. Potential Process Benefits

Besides new challenges, though, computational plenty will enable new and more powerful process-related approaches. It will enable new and more powerful self-monitoring software and computing via on-chip co-processors for assertion checking, trend analysis, intrusion detection, or verifying proof-carrying code. It will enable higher levels of abstraction, such as pattern-oriented programming, multi-aspect oriented programming, domain-oriented visual component assembly, and programming by example with expert feedback on missing portions. It will enable simpler brute-force solutions such as exhaustive case evaluation vs. complex logic.

It will also enable more powerful software, hardware, human factors, and systems engineering tools that provide feedback to developers based on domain knowledge, construction knowledge, human factors knowledge, systems engineering knowledge, or management knowledge. It will enable the equivalent of seat belts and air bags for user-programmers. It will support show-and-tell documentation and much more powerful system query and data mining techniques. It will support realistic virtual game-oriented systems and software engineering education and training. On balance, the added benefits of computational plenty should significantly outweigh the added challenges.

2.9 Wild Cards: Autonomy and Bio-Computing

"Autonomy" covers technology advancements that use computational plenty to enable computers and software to autonomously evaluate situations and determine best-possible courses of action. Examples include:

- Cooperative intelligent agents that assess situations, analyze trends, and cooperatively negotiate to determine best available courses of action.

- Autonomic software that uses adaptive control techniques to reconfigure itself to cope with changing situations.
- Machine learning techniques that construct and test alternative situation models and converge on versions of models that will best guide system behavior.
- Extensions of robots at conventional-to-nanotechnology scales empowered with autonomy capabilities such as the above.

Combinations of biology and computing include:

- Biology-based computing, which uses biological or molecular phenomena to solve computational problems beyond the reach of silicon-based technology.
- Computing-based enhancement of human physical or mental capabilities, perhaps embedded in or attached to human bodies or serving as alternate robotic hosts for (portions of) human bodies.

Examples of books describing these capabilities are Kurzweil's *The Age of Spiritual Machines* [Kurzweil, 1999] and Drexler's books *Engines of Creation* and *Unbounding the Future: The Nanotechnology Revolution* [Drexler, 1986, Drexler, Peterson, and Pergamit, 1991]. They identify major benefits that can potentially be derived from such capabilities, such as artificial labor, human shortfall compensation (the five senses, healing, life span, and new capabilities for enjoyment or self-actualization), adaptive control of the environment, or redesigning the world to avoid current problems and create new opportunities.

On the other hand, these books and other sources such as Dyson's *Darwin Among the Machines: The Evolution of Global Intelligence* [Dyson, 1997] and Joy's article "Why the Future Doesn't Need Us" [Joy, 2000], and Crichton's bio/nanotechnology novel *Prey* [Crichton, 2002], identify major failure modes that can result from attempts to redesign the world, such as loss of human primacy over computers, overempowerment of humans, and irreversible effects such as plagues or biological dominance of artificial species. From a software process standpoint, processes will be needed to cope with autonomy software failure modes such as undebuggable self-modified software, adaptive control instability, interacting agent commitments with unintended consequences, and commonsense reasoning failures.

As discussed in Dreyfus and Dreyfus' *Mind Over Machine* [Dreyfus and Dreyfus, 1986], the track record of artificial intelligence predictions shows that it is easy to overestimate the rate of AI progress. But a good deal of AI technology is usefully at work today and, as we have seen with the Internet and World Wide Web, it is also easy to underestimate rates of IT progress as well. It is likely that the more ambitious predictions above will not take place by 2025, but it is more important to keep both the positive and negative potentials in mind in risk-driven experimentation with emerging capabilities in these wild-card areas between now and 2025.

3. A SCALABLE SPIRAL PROCESS MODEL FOR 21ST CENTURY SYSTEMS AND SOFTWARE

3.1. 21st Century System and Software Development and Evolution Modes

In the next 10–20 years, several 21st century system and software development and evolution modes will have emerged as the most cost-effective ways to develop needed capabilities in the context of the trends discussed in Section 2. The four most common modes are likely to be exploratory development of unprecedented capabilities, business model-based user programming, hardware and software product lines, and network-centric systems of systems. Each is discussed below, along with the primary processes that will most likely best fit their situations.

Exploratory development processes will continue to be used for new products in mainstream organizations and in new areas such as nanotechnology, advanced biotechnology and robotics, virtual reality, and cooperative agent-based systems. They will still focus on highly flexible processes for skill-intensive rapid prototyping. But pressures for rapid transition from prototype to fielded product will increase the emphasis on the concept development phase to meet criteria for demonstrating that the new concepts can be made sufficiently robust, scalable, and cost-effectively producible. The process and associated product capabilities will also need to be selectively open in order to support open-innovation forms of collaborative development with other companies providing complementary capabilities [Chesbrough, 2003].

Business model-based user programming will expand its scope to continue to address the need to produce more and more software capabilities by enabling them to be produced directly by users, as with spreadsheet programs, computer-aided design and manufacturing (CAD/CAM), and website development and evolution. Much of the expanded scope will be provided by better-integrated and more tailorable Enterprise Resource Planning (ERP) COTS packages. As discussed in Section 2.8.1, computational plenty and increased domain understanding will enable more powerful,

safer, and easier-to-use user programming capabilities such as programming-by-example with expert-system feedback on missing portions. Larger extensions to the ERP framework may be carried out by in-house software development, but specialty-houses with product-line-based solutions will become an increasingly attractive outsourcing solution.

Hardware and software product lines on the hardware side will increasingly include product lines for transportation, communications, medical, construction, and other equipment. On the software side, they will increasingly include product lines for business services, public services, and information infrastructure. Compared to current product lines in these areas, the biggest challenges will be the increasing rates of change and decreasing half-lives of product line architectures, and the increasing proliferation of product line variabilities caused by globalization.

Network-centric systems of systems. As discussed in Section 2.6, similar challenges are being faced by organization in the process of transforming themselves from collections of weakly coordinated, vertically integrated stovepipe systems into seamlessly interoperable network-centric systems of systems (NCSOS). The architectures of these NCSOS are highly software-intensive and, as with the product line architectures above, need to be simultaneously robust, scalable, and evolvable in flexible but controllable ways. Both product lines and NCSOS thus need processes such as the Internet spiral described in Section 2.4.1, but due to competitive pressures, their processes must generally operate on much tighter timescales than were involved in the early evolution of the Internet. In Sections 3.2 and 3.3, we describe an emerging scalable spiral process model for developing and evolving 21st century product lines and NCSOS.

3.2. Overview of the Scalable Spiral Process Model

Based on our experiences in adapting the spiral model to the development of software-intensive systems of systems representative of the 21st century trends discussed above, we have been converging on a scalable spiral process model that has shown in partial implementation to date to scale well from small e-services applications to superlarge defense systems of systems, and multienterprise supply chain management systems. The model contains familiar elements, but organizes them in ways that involve new approaches for most current enterprises to enterprise organization, contracting, incentives, staffing, education, and career development.

Figure 5 shows a single increment of the development and evolution portion of the model. It assumes that the organization has developed:

- A best-effort definition of the system's steady-state capability.
- An incremental sequence of prioritized capabilities culminating in the steady-state capability.
- A Feasibility Rationale providing sufficient evidence that the system architecture will support the incremental capabilities, that each increment can be developed within its available budget and schedule, and that the series of increments create a satisfactory return on investment for the organization and mutually satisfactory outcomes for the success-critical stakeholders.
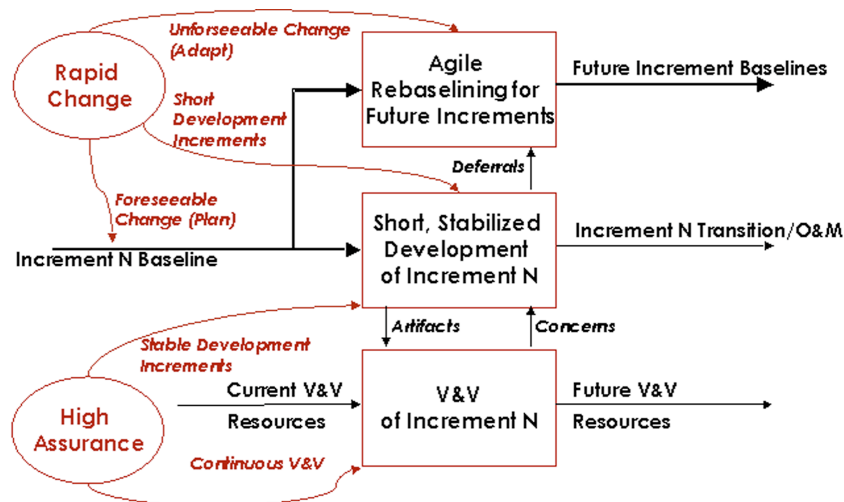

Figure 5. The Scalable Spiral Process Model: Increment activities.

Techniques for achieving these front-end outcomes and their associated anchor-point milestones are provided in Boehm [1988, 1996], Boehm et al. [2004], Boehm and Port [2001], Kroll and Kruchten [2003], and Royce [1998].

As seen in Figure 5, the model is organized to simultaneously address the conflicting 21st century challenges of rapid change and high assurance of dependability. It also addresses the need for rapid fielding of incremental capabilities with a minimum of rework, and the other major 21st century trends involving integration of systems and software engineering, COTS components, legacy systems, globalization, and user value considerations discussed above.

In *Balancing Agility and Discipline* [Boehm and Turner, 2004], we found that rapid change comes in two primary forms. One is relatively predictable change that can be handled by the plan-driven Parnas strategy [Parnas, 1979] of encapsulating sources of change within modules, so that the effects of changes are largely confined to individual modules. The other is relatively unpredictable change that may appear simple (such as adding a "cancel" or "undo" capability [Bass and John, 2003]), but often requires a great deal of agile adaptability to rebaseline the architecture and incremental capabilities into a feasible solution set.

The need to deliver high-assurance incremental capabilities on short fixed schedules means that each increment needs to be kept as stable as possible. This is particularly the case for very large systems of systems with deep supplier hierarchies (often 6–12 levels), in which a high level of rebaselining traffic can easily lead to chaos. In keeping with the use of the spiral model as a risk-driven process model generator, the risks of des-

tabilizing the development process make this portion of the project into a waterfall-like build-to-specification subset of the spiral model activities. The need for high assurance of each increment also makes it cost-effective to invest in a team of appropriately skilled personnel to continuously verify and validate the increment as it is being developed.

However, "deferring the change traffic" does not imply deferring its change impact analysis, change negotiation, and rebaselining until the beginning of the next increment. With a single development team and rapid rates of change, this would require a team optimized to develop to stable plans and specifications to spend much of the next increment's scarce calendar time performing tasks much better suited to agile teams.

3.2.1. Agile Rebaselining and the C2ISR Metaphor

The appropriate metaphor for these tasks is not a build-to-specification metaphor or a purchasing-agent metaphor but an adaptive "command-control-intelligence-surveillance-reconnaissance" (C2ISR) metaphor. It involves an agile team performing the first three activities of the C2ISR "Observe, Orient, Decide, Act" (OODA) loop for the next increments, while the plan-driven development team is performing the "Act" activity for the current increment. "Observing" involves monitoring changes in relevant technology and COTS products, in the competitive marketplace, in external interoperating systems and in the environment; and monitoring progress on the current increment to identify slowdowns and likely scope deferrals. "Orienting" involves performing change impact analyses, risk analyses, and tradeoff analyses to assess candidate rebaselining op-
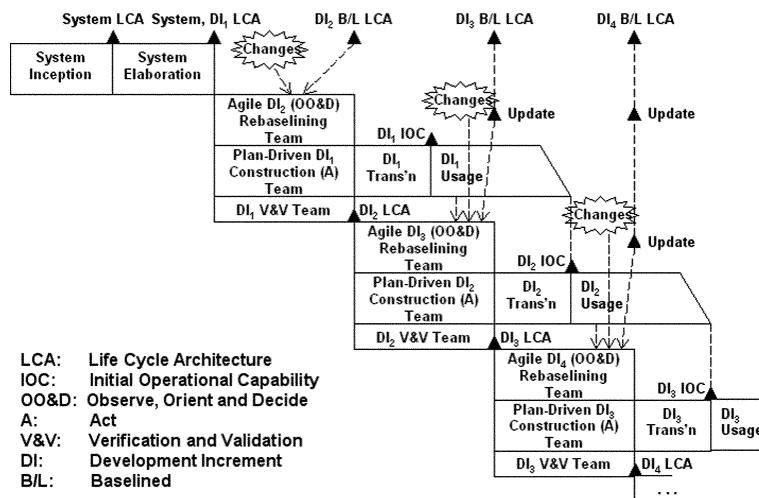

Figure 6. The Scalable Spiral Process Model: Life cycle view.

tions for the upcoming increments. "Deciding" involves stakeholder renegotiation of the content of upcoming increments, architecture rebaselining, and the degree of COTS upgrading to be done to prepare for the next increment. It also involves updating the future increments' Feasibility Rationales to ensure that their renegotiated scopes and solutions can be achieved within their budgets and schedules.

A successful rebaseline means that the plan-driven development team can hit the ground running at the beginning of the "Act" phase of developing the next increment, and the agile team can hit the ground running on rebaselining definitions of the increments beyond. Figure 6 shows how this three-team cycle (lean, plan-driven, stabilized developers; thorough V&Vers; agile, pro-active rebaseliners) plays out from one increment to the next, including the early product line or systems-of-systems Inception and Elaboration phases with their pass-fail Life Cycle Objectives and Life Cycle Architecture exit milestones.{FIG6} Note that "(OO&D)" in each agile rebaselining increment stands for "Observe, Orient, and Decide" and not "Object Oriented Design". The (A) below it stands for the "Act" portion of the OODA loop for the current increment. Note also that, as much as possible, usage feedback from the previous increment is not allowed to destabilize the current increment, but is fed into the definition of the following increment. Of course, some level of mission-critical updates will need to be fed into the current increment, but only when the risk of not doing so is greater that the risk of destabilizing the current increment.

Again, in keeping with the use of the spiral model as a risk-driven process model generator, the risks of not adequately and rapidly adapting the next increments to address the sources of needed change mean that this portion of the project is not a build-to-specification waterfall process. Instead, it is a risk-driven set of concurrent prototyping, analysis, and stakeholder renegotiation activities leading to a best-possible redefinition of the plans and specifications to be used by the stabilized development team for the next increment.

3.2. Synchronizing Hardware, Software, and Human Systems Processes

The scalable spiral process shown in Figures 5 and 6, and its associated LCO, LCA, and IOC phase gates, provide specificity to the types of 21st century chaordic and adaptive processes discussed in Hock's *Birth of the Chaordic Age* [Hock, 1999] and Highsmith's *Adaptive Software Development* [Highsmith, 2000]. The partial implementations of the process so far have been a straightforward fit to software-intensive systems. But

the process also needs further interpretations to cope with the realties of integrating the hardware, software, and human systems elements of complex systems, and especially systems of systems with deep supplier hierarchies.

As discussed by Rechtin and Maier [1997], hardware manufacturing may have different time phasing than software, given its need for procurement of long-lead components and production-line piloting, tooling, and preparation. The risks of getting the details of the production line wrong are often (but not always) much higher than the risks of getting the detailed design of the software wrong. This means that the hardware and embedded software elements need a top-to-bottom Critical Design Review before proceeding into production, while the nonembedded software detailed design can generally proceed with individual inspections as each software component's detailed design is prepared.

This often means that several software increments may be fielded between major hardware increments. In general, though, hardware-intensive organizations that have used the LCO and LCA milestones and their Feasibility Rationale pass/fail criteria to synchronize hardware and software architecture definition have found this highly advantageous. Further, 21st century trends toward flexible software-driven manufacturing, digital product representations, and "3D hardcopy" [Rosenbloom, 2005] prototyping of digitally designed components and products continue to reduce but not eliminate the differences between hardware and software processes.

3.2.1. Human Systems Solutions and Challenges

Similar challenges may affect the nature of human systems solutions. Labor-management negotiations may impose additional constraints on the pace of business process reengineering, and the costs and risks of frequently retraining users and operators may cause user-increment releases to consist of several stabilized software development increments. Here again, though, the macro-level LCO, LCA, and IOC milestones have proved valuable in keeping the human systems increments synchronized with the hardware and software increments. Checkland's soft-systems methodology [Checkland, 1999] and the recent *Handbook on Human Systems Integration* [Booher, 2003] are excellent sources of guidance on these issues.

Further human systems challenges involve the people in the plan-driven development, V&V, and agile rebaselining teams. The budgets and schedules for the development team must allow for some level of their participation in the agile rebaselining process. This is

necessary both because they are success-critical stakeholders who need to be represented, and because they have essential expertise for determining feasible rebaselining solutions. Further, their budgets and schedules must also allow for some time to perform mission-critical updates of the currently operating system, as discussed above.

Another reality is that these effects will not impact the various development teams uniformly. Thus, some slack must be incorporated into the process to keep the ensemble of feature teams synchronized and stabilized. The use of continuous builds and sub-increment slack periods used by Microsoft [Cusumano and Selby, 1996] is a good example of this practice. And in general, the need for some slack to keep development teams productive is well covered in DeMarco's *Slack* [DeMarco, 2001].

This surfaces the most critical success factors of all: recruiting, nurturing, and sustaining the scarce people with the right combinations of management and technical skills to keep the three teams and the other success-critical stakeholders balanced, collaborative, and focused on developing the most successful system possible. This involves many teambuilding, balancing, packaging, and negotiation skills. Teambuilding includes getting key stakeholder representatives to participate and buy into the process and shared vision for the system. Balancing includes not only agility and discipline, but also rebalancing the size of the development, V&V, and agile rebaselining teams as the need for their capabilities evolves across the life cycle. Packaging and negotiation involve the ability to create new technical and/or management options to keep all of the stakeholders positively motivated, even though the changes may reduce their scope and resource levels.

3.4. Model Experience to Date

The scalable spiral model has been evolving with experience and has not yet been fully implemented on a large, completed project. However, its principles and practices build on many successful project experiences and unsuccessful project lessons learned. Specific examples of projects that have successfully balanced agile and plan-driven methods are the agile-based Thought-Works lease management project [Elssamadisy and Schalliol, 2002] and the plan-based CCPDS-R project [Royce, 1998; see also Section 4.1]. More generally, Collins' book *Good to Great* [Collins, 2001] identifies 11 companies with exceptional performance records as having successfully transformed themselves into simultaneously having a strong Ethic of Entrepreneurship and a strong Culture of Discipline.

The use of concurrent independent verification and validation teams has been successfully practiced and evolved since the 1970s [Rubey, Dana, and Biché, 1979]. More recent successful continuous V&V practices include the continuous build practices at Microsoft [Cusumano and Selby, 1996] and in agile methods [Beck, 1999]. Pro-active investments in agile next-increment teams are successfully used in exploiting disruptive technologies at companies such as Hewlett Packard, Seagate, and Johnson and Johnson [Christensen, 2000], and in practicing open innovation in companies such as HP, IBM, Intel, and Lucent [Chesbrough, 2003]. Successful use of the anchor point milestones and evolutionary development using the Rational Unified Process [Rational, 2001] and the WinWin Spiral model [Boehm et al., 1998] has been experienced on numerous small, medium, and large software projects, and on hardware projects at such companies as Xerox and Johnson and Johnson. Partial implementations of the model also providing improvement are being evolved on the very large scale U.S. Army Future Combat systems program, large space systems, and commercial supply chain systems.

Experiences to date indicate that the three teams' activities are not as neatly orthogonal as they look in Figures 5 and 6. Feedback on development shortfalls from the V&V team either requires a response from the development team (early fixes will be less disruptive and expensive than later fixes), or deferral to a later increment, adding work and coordination by the agile team. The agile team's analyses and prototypes addressing how to accommodate changes and deferred capabilities need to draw on the experience and expertise of the plan-driven development team, requiring some additional development team resources and calendar time. Additional challenges arise if different versions of each increment are going to be deployed in different ways into different environments. The model has sufficient degrees of freedom to address such challenges, but they need to be planned for within the project's schedules and budgets.

4. IMPLICATIONS FOR 21ST CENTURY ENTERPRISE PROCESSES

In working with our commercial and aerospace Affiliates on how they can best evolve to succeed as 21st century enterprises, we have found several 20th century process-related institutions that need to be significantly rethought and reworked to contribute to success. We will discuss two leading examples below: acquisition practices and human relations. In the interest of brevity, some other important institutions needing rethinking

and rework but not discussed in detail are continuous process improvement (repeatability and optimization around the past vs. adaptability and optimization around the future); supplier management (adversarial win-lose vs. team-oriented win-win); internal R&D strategies (core capability research plus external technology experimentation vs. full-spectrum self-invention); and enterprise integration (not-invented-here stovepipes vs. enterprise-wide learning and sharing).

4.1 Acquisition as C2ISR vs. Purchasing

The 20th century purchasing agent or contracts manager is most comfortable with a fixed procurement to a set of prespecified requirements; selection of the least-cost, technically adequate supplier; and a minimum of bothersome requirements changes. Many of our current acquisition institutions—regulations, specifications, standards, contract types, award fee structures, reviews and audits—are optimized around this procurement model.

Such institutions have been the bane of many projects attempting to deliver successful systems in a world of emerging requirements and rapid change. The project people may put together good technical and management strategies to do concurrent problem and solution definition, team building, and mutual-learning prototypes and options analyses. Then they find that their progress payments and award fees involve early delivery of complete functional and performance specifications. Given the choice between following their original strategies and getting paid, they proceed to marry themselves in haste to a set of premature requirements, and find themselves repenting at leisure for the rest of the project (if any leisure time is available).

Build-to-specification contract mechanisms still have their place, but it is just for the stabilized increment team in Figure 5. If such mechanisms are applied to the agile rebaselining teams, frustration and chaos ensues. What is needed for the three-team approach is separate contracting mechanisms for the three team functions, under an overall contract structure that enables them to be synchronized and rebalanced across the life cycle. Also needed are source selection mechanisms more likely to choose the most competent supplier, using such approaches as competitive exercises to develop representative system artifacts using the people, products, processes, methods, and tools in the offeror's proposal.

A good transitional role model is the CCPDS-R project described in Royce [1998]. Its U.S. Air Force customer and TRW contractor (selected using a competitive exercise such as the one described above) reinterpreted the traditional defense regulations, specifications, and standards. They held a Preliminary Design Review: not a PowerPoint show at Month 4, but a fully validated architecture and demonstration of the working high-risk user interface and networking capabilities at Month 14. The resulting system delivery, including over a million lines of software source code, exceeded customer expectations within budget and schedule.

Other good acquisition approaches are the Scandinavian Participatory Design approach [Ehn, 1990], Checkland's Soft Systems Methodology [Checkland, 1999], lean acquisition and development processes [Womack and Jones, 1996], and Shared Destiny-related contracting mechanisms and award fee structures [Deck, Strom, and Schwartz, 2001; Rational, 2001]. These all reflect the treatment of acquisition using a C2ISR metaphor rather than a purchasing-agent metaphor.

4.2. Human Relations

Traditional 20th century human relations or personnel organizations and processes tend to emphasize individual vs. team-oriented reward structures and monolithic career paths. These do not fit well with the team-oriented, diverse-skill needs required for 21st century success.

In *Balancing Agility and Discipline* [Boehm and Turner, 2004], we found that plan-oriented people are drawn toward organizations that thrive on order. People there feel comfortable and empowered if there are clear policies and procedures defining how to succeed. On the other hand, agility people are drawn toward organizations that thrive on chaos. People there feel comfortable and empowered if they have few policies and procedures, and many degrees of freedom to determine how to succeed. In our USC Balancing Agility and Discipline Workshops, we found that most of our Affiliates had cultures that were strongly oriented toward one of these poles, with the challenge of evolving toward the other pole without losing the good features of their existing culture and staff.

The three-team approach provides a way for organizations to develop multiple role-oriented real or virtual skill centers with incentive structures and career paths focused both on excellence within one's preferred role and teamwork with the other contributors to success. Some other key considerations are the need for some rotation of people across team roles or as part of integrated product teams to avoid overspecialization, and the continual lookout for people who are good at all three team roles and are strong candidates for project-level or organization-level management or technical leadership careers.

A good framework for pursuing a human relations strategy for 21st century success is the People Capability Maturity Model [Curtis, Hefley, and Miller, 2002]. Its process areas on participatory culture, workgroup development, competency development, career development, empowered workgroups, and continuous workforce innovation emphasize the types of initiatives necessary to empower people and organizations (such as the purchasing agents and departments discussed above) to cope with the challenges of 21st century system development. The P-CMM book also has several case studies of the benefits realized by organizations adopting the model for their human relations activities. The Collins *Good to Great* book [Collins, 2001] is organized around a stepwise approach characterizing the 11 outstanding performance companies' transformation into cultures having both an ethic of entrepreneurship and culture of discipline. It begins with getting the right people and includes setting ambitious but achievable goals and constancy of purpose in achieving them.

5. CONCLUSIONS

The surprise-free and wild-card 21st century trends discussed in Section 2 provide evidence that significant changes in and integration of systems and software engineering processes will be needed for successful 21st century enterprises. Particularly important are changes that emphasize value generation and enable dynamic balancing of the agility, discipline, and scalability necessary to cope with the 21st century challenges of increasing rapid change, high dependability, and scalability to globally-integrated, software-intensive systems of systems.

Section 3 presents an emerging scalable spiral process framework and set of product and process strategies for coping with these challenges. They are proving to be effective as we evolve them with our industry and government Affiliate organizations. The product strategies involve system and software architectures that encapsulate sources of rapid unpredictable change into elements developed by agile teams within a framework and set of elements developed by a plan-driven team as shown in Figure 2. The process strategies involve stabilized increment development executed by the plan-driven team and verified and validated by a V&V team, along with concurrent agile, pro-active change assessment and renegotiation of stable plans and specifications for executing the next increment, as shown in Figures 5 and 6.

The scalable spiral process is not fully tested in practice on large systems. But it is founded on experi-ence-based practices, and has demonstrated improvements in ongoing projects. And there is a dearth of scalable alternative processes for dealing with high rates of unanticipated change. Agile methods do not scale up well, and plan-driven methods encounter serious problems of excess rework in adapting to change, or of product obsolescence in declining to adapt to change.

However, process and architecture strategies are only necessary and not sufficient conditions for enterprise success. Section 4 identifies and describes some of the complementary organizational initiatives that will be necessary to enable the product and process strategies to succeed. These include rethinking and reworking acquisition contracting practices, human relations, continuous process improvement, supplier management, internal R&D strategies, and enterprise integration.

REFERENCES

C. Albert and L. Brownsword, Evolutionary process for integrating COTS-based systems (EPIC): An overview, CMU/SEI-20030TR-009, Software Engineering Institute, Pittsburgh, PA, 2002.

G. Anthes, The future of IT, Computerworld (March 7, 2005), 27–36.

W.B. Arthur, Increasing returns and the new world of business, Harvard Bus Rev (July/August 1996), 100–109.

V. Basili, M. Zelkowitz, F. McGarry, J. Page, S. Waligora, and R. Pajerski, Special Report: SEL's process-improvement program, Software (November 1995), 83–87.

L. Bass and B.E. John, Linking usability to software architecture patterns through general scenarios, J Syst Software 66(3) (2003), 187–197.

K. Beck, Extreme programming explained, Addison Wesley, Reading, MA, 1999.

S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Gruenbacher (Editors), Value-based software engineering, Springer, New York, 2005.

B. Boehm, Software engineering economics, Prentice Hall, Englewood Cliffs, NJ, 1981.

B. Boehm, A spiral model for software development and enhancement, Computer (May 1988), 61–72.

B. Boehm, Anchoring the software process, Software (July 1996), 73–82.

B. Boehm and W. Hansen, The spiral model as a tool for evolutionary acquisition, CrossTalk (May 2001), 4–11.

B. Boehm and C.A.R. Hoare (Editors), Proceedings, 1975 International Conference on Reliable Software, ACM/IEEE, New York, April 1975.

B. Boehm and A. Jain, "An initial theory of value-based software engineering," Value-based software engineering, A. Aurum, S. Biffl, B. Boehm, H. Erdogmus, and P. Gruenbacher (Editors), Springer, New York, 2005.

B. Boehm and D. Port, Balancing discipline and flexibility with the spiral model and MBASE, CrossTalk (December 2001), 23–28.

B. Boehm and R. Turner, Balancing agility and discipline, Addison Wesley, Reading, MA, 2004.

B. Boehm, A.W. Brown, V. Basili, and R. Turner, Spiral acquisition of software-intensive systems of systems, CrossTalk (May 2004), 4–9.

B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy, Using the WinWin Spiral Model: A case study, IEEE Comput (July 1998), 33–44.

H. Booher.(Editor), Handbook of human systems integration, Wiley, New York, 2003.

F. Brooks, The mythical man-month (2nd edition), Addison Wesley, Reading, MA, 1995.

P. Checkland, Systems thinking, systems practice (2nd edition), Wiley, New York, 1999.

H. Chesbrough, Open innovation, Harvard Business School Press, Cambridge, MA, 2003.

M.B. Chrissis, M. Konrad, and S. Shrum, CMMI, Addison Wesley, Reading,MA, 2003.

C. Christensen, The innovator's dilemma, Harvard Business School Press, Cambridge, MA, 1997 and Harper Collins, New York, 2000.

A. Cockburn, Agile software development, Addison Wesley, Reaing, MA, 2002.

J. Collins, Good to great, Harper Collins, New York, 2001.

D. Crawford, Editorial pointers, Commun ACM (October 2001), 5.

M. Crichton, Prey, Harper Collins, New York, 2002.

B. Curtis, B. Hefley, and S. Miller, The people capability maturity model, Addison Wesley, Reading, MA, 2002.

M. Cusumano and R. Selby, Microsoft secrets, Harper Collins, New York, 1996.

M. Deck, M. Strom, and K. Schwartz, The emerging model of co-development, Res Technol Management (December 2001).

T. DeMarco, Slack, Broadway Books, 2001.

E. Dijkstra, Panel remarks, Software Engineering: As It Should Be, ICSE 4 (September 1979), see also EWD 791 at http://www.cs.utexas/users/EWD.

E.K. Drexler, Engines of creation, Anchor Press, 1986.

K.E. Drexler, C. Peterson, and G. Pergamit, Unbounding the future: The nanotechnology revolution, William Morrow, 1991.

H. Dreyfus and S. Dreyfus, Mind over machine, Macmillan, New York, 1986.

L. Druffel, N. Loy, R. Rosenberg, R. Sylvester, and R. Volz, Information architectures that enhance operational capability in peacetime and warfare, USAF-SAB Study Report, U.S. Air Force, Washington, DC, February 1994.

G.B. Dyson, Darwin among the machines: The evolution of global intelligence, Helix Books/Addison Wesley, Reading, MA, 1997.

P. Ehn (Editor), Work-oriented design of computer artifacts, Earlbaum, 1990.

A. Elssamadisy and G. Schalliol, Recognizing and responding to "bad smells" in extreme programming, Proc Twenty-Fourth Int Conf Software Eng (ICSE2002), Orlando, FL, May 2002, pp. 617–622.

FCIO (Federal CIO Council), A practical guide to federal enterprise architecture, Version 1.0, FCIO, Washington, DC, February 2001.

A. Fuggetta, "Software process: A roadmap," The future of software engineering, A. Finkelstein (Editor), ACM Press, 2000.

P. Gerrard and N. Thompson, Risk-based e-business testing, Artech House, 2002.

R. Grady, Successful software process improvement, Prentice Hall, Englewood Cliffs, NJ, 1997.

P. Gruenbacher, S. Koeszegi, and S. Biffl, "Stakeholder value proposition elicitation and reconciliation," Value-Based Software Engineering, A. Aurum, S. Biffl, B. Boehm, H. Erdogmus, and P. Gruenbacher (Editors), Springer, New York, 2005.

D. Harned and J. Lundquist, What transformation means for the defense industry, McKinsey Quart (November 3, 2003), 57–63.

J. Highsmith, Adaptive software development, Dorset House, 2000.

D. Hock, Birth of the Chaordic Age, Berrett-Koehler, 1999.

G. Hofstede, Culture and organizations, McGraw Hill, New York, 1997.

L. Huang, A value-based process achieving software dependability, Proc Software Process, Workshop 2005, May 2005.

W. Humphrey, Introduction to the personal software process, Addison Wesley, Reading, MA, 1997.

W. Humphrey: Introduction to the team software process, Addison Wesley, Reading, MA, 2000.

INCOSE, Systems engineering technical vision, Draft Version 1.5, INCOSE, June 2005.

ISO (International Standards Organization), Standard for information technology—software life cycle processes, ISO/IEC 12207, ISO, 1995.

ISO (International Standards Organization), Systems engineering—system life cycle processes, ISO/IEC 15288, ISO, 2002.

B. Joy, Why the future doesn't need us, Wired (April 2000).

L. Koskela and L. Howell, The underlying theory of project management is obsolete, Proc PMI Res Conf, 2002, pp. 293–302.

P. Kroll and P. Kruchten, The rational unified process made easy: A practitioner's guide to the rational unified process, Addison Wesley, Reading, MA, 2003.

R. Kurzweil, The age of spiritual machines, Penguin, New York, 1999.

A. Maslow, Motivation and personality, Harper and Row, New York, 1954.

G. Moore, Inside the tornado, Harper Collins, New York, 1995.

J. Musa, Software reliability engineering, McGraw Hill, New York, 1999.

D. Parnas, Designing software for ease of extension and contraction, Trans Software Eng IEEE SE-5, 1979.

D. Patterson, 20th century vs. 21st century C&C: The SPUR manifesto, Commun ACM (March 2005), 15–16.

M. Paulk, C. Weber, B. Curtis, and M. Chrissis, The Capability Maturity Model, Addison Wesley, Reading, MA, 1994.

M. Phongpaibul and B. Boehm, Improving quality through software process improvement in Thailand: Initial analysis, Proc ICSE 2005 Workshop Software Qual, May 2005.

PITAC (President's Information Technology Advisory Committee), Report to the President: Information technology research: Investing in our future, PITAC, Washington, DC, 1999.

J. Putman, Architecting with RM-ODP, Prentice Hall, Englewood, Cliffs, NJ, 2001.

Rational, Inc., Driving better business with better software economics, Rational Software Corp (now part of IBM), 2001.

E. Rechtin, Systems architecting, Prentice Hall, Englewood Cliffs, NJ, 1991.

E. Rechtin and M. Maier, The art of systems architecting, CRC Press, Boca Raton, FL, 1997.

D. Reifer and B. Boehm, A model contract/subcontract award fee plan for large, change-intensive software acquisitions, USC Center for Software Engineering, University of Southern California, Los Angeles, April 2003.

A. Rosenbloom, 3D hard copy, Commun ACM (June 2005), 32–35.

W.E. Royce, Software project management, Addison Wesley, Reading, MA, 1998.

J.R. Rubey, J.A. Dana, and P.W. Biché, Quantitative aspects of software validation, IEEE Trans SW Eng (June 1975), 150–155.

Standish Group, Extreme chaos, http://www.standishgroup.com, 2001.

S. Toulmin, Cosmopolis, University of Chicago Press, Chicago, IL, 1992.

U.S. Department of Defense (USD), MIL-STD-1521B: Reviews and audits, USD, Washington, DC, 1985.

U.S. Department of Defense (USD), DOD-STD-2167A: Defense system software development, USD, Washington, DC, 1988.

J. Womack and D. Jones, Lean thinking: Banish waste and create wealth in your corporation, Simon & Schuster, New York, 1996.

J.P. Womack, D.T. Jones, and D. Roos, The machine that changed the world: The story of lean production, Harper Perennial, New York, 1990.

Y. Yang, J. Bhuta, D. Port, and B. Boehm, Value-based processes for COTS-based applications, IEEE Software (July/August 2005), 54–62.

Y. Yang, B. Boehm, and D. Port, A contextualized study of COTS-based e-service projects, Proc Software Process Workshop 2005, Springer, New York, 2005, to appear.

J. Zachman, A framework for information systems architecture, IBM Syst J (1987).

Barry Boehm is the TRW Professor of Software Engineering in the Computer Science and Industrial and Systems Engineering Departments at the University of Southern California. He is also Director of the USC Center for Software Engineering. He has a BA from Harvard (1957), and an MA (1961) and PhD (1964) from UCLA, all in mathematics, and an honorary ScD (2000) in Computer Science from the University of Massachusetts. He served within the U.S. Department of Defense (DoD) from 1989 to 1992 as director of the DARPA Information Science and Technology Office and as director of the DDR&E Software and Computer Technology Office. At DARPA, he was responsible for the acquisition of over $1 billion in advanced information technology, including DARPA's contributions to the ARPAnet/Internet, artificial intelligence, high performance computing, robotics, software technology, and virtual reality simulation. He worked at TRW from 1973 to 1989, serving as advanced technology business area manager, division chief engineer, and chief scientist of the Defense Systems Group. At the Rand Corporation from 1959 to 1973, he worked in software and systems analysis, including leadership of the major Air Force CCIP-85 (Command and Control Information Processing-1985) mission analysis in 1971–72, and as head of the Information Sciences Department. He was a programmer-analyst at General Dynamics from 1955 to 1959. His current research interests involve integrating systems and software engineering into a value-based framework, including processes, methods, and tools for value-based systems and software definition, architecting, development, and validation. His contributions to the field include the Constructive Cost Model (COCOMO), the Constructive Systems Engineering Cost Model (COSYSMO), the Spiral Model of the systems and software development and evolution process, and the Theory W (win-win) approach to systems and software management and requirements determination. He is a Fellow of the primary professional societies in computing (ACM), aerospace (AIAA), electronics (IEEE), and systems engineering (INCOSE), and a member of the U.S. National Academy of Engineering.