Spiral Acquisition of Software-Intensive Systems of Systems

Dr. Barry Boehm and A. Winsor Brown
University of Southern California

Dr. Victor Basili
University of Maryland

Dr. Richard Turner
George Washington University and OSD/Software-Intensive Systems

Monday, 19 April 2004
Track 2: 4:30 - 5:15
Ballroom B

The Department of Defense and other organizations are finding that the acquisition and evolution of complex systems of systems is both software-intensive and fraught with old and new sources of risk. This article summarizes both old and new sources of risk encountered in acquiring and developing complex software-intensive systems of systems. It shows how these risks can be addressed via risk analysis, risk management planning and control, and application of the risk-driven Win-Win Spiral Model. It will also discuss techniques for handling complicating factors such as compound risks, incremental development, and rapid change, and illustrates the use of principles and practices with experience in applying the model to the U.S. Army Future Combat Systems program and similar programs.

Current trends toward the transformation of warfare (and other large-scale competitive pursuits) into network-centric and knowledge-based systems of systems show great promise for competitive advantages over traditionally organized groups of largely independent components. However, these transformational systems of systems are critically dependent on the successful functioning of their computer software [1, 2]. This article summarizes the benefits provided by software for such transformational systems and identifies a top-10 list of risks and challenges that need to be resolved in developing and evolving software-intensive systems. It then briefly summarizes the Win-Win Spiral Model described in more detail in CROSSTALK [3, 4], and shows how its application can be used to mitigate the top-10 software-intensive system risks and challenges.

Software Benefits for Transformational Systems

While simpler, tangible hardware configurations are easier to manage than software-intensive system acquisitions and operations, pure hardware-based solutions cannot provide several key benefits that software can provide for complex transformational systems of systems. These include the following:

- **The need to accommodate many combinations of mission options.** Trying to accommodate these in hardware leads to earlier freezing of option choices, more complex hardware production, inflexible human computer interfaces, and very expensive and time-consuming hardware field upgrades.
- **The need for rapid response to change.** The pace of unforeseeable change continues to accelerate. Accommodating such change runs into much the same set of hardware difficulties and software opportunities as does the accommodation of many options. Also, many sources of software change are often accommodated by commercial off-the-shelf (COTS) software upgrades made by vendors who need to stay competitive.
- **The need for fielding partial capabilities.** Some options for doing this in hardware are available, but again with higher needs to pre-commit to interface choices. Current Department of Defense (DoD) policies that emphasize evolutionary acquisition [5, 6] are much easier to accommodate via simpler hardware platforms and evolutionary software upgrades.

Table 1: *Average-Case Software Development Time versus Size*

Size (KSLOC)	300	1,000	3,000	10,000
Time (Months)	33	50	72	108

Risks and Challenges

The transformational benefits that software capabilities provide are compelling but come with associated risks and challenges. The ability to *accommodate many combinations of mission options* comes with the need for more software and longer delivery time for software-intensive system of systems (SISOS). A large SISOS such as the national air traffic control system or a major integrated industrial manufacturing and supply chain management system will have more than 10 million source lines of code (or 10,000 KSLOC) that need to be developed and integrated.

SISOS managers are frequently surprised when the first cost-schedule estimation model run indicates that software with 10,000 KSLOC will take at least nine years to develop using traditional methods. Most software cost-schedule models have calibrated relationships indicating that the calendar time, in months, required for average-case software development scales roughly as five times the cube root of the size in KSLOC (see Table 1), or roughly,

Average Case Development Time = 5 x Cube Root (KSLOC)

Clearly, if the SISOS is needed quickly, replacements for traditional software development methods are needed. These go from processes enabling more concurrent development to acquisition methods that are both less bureaucratic and more able to control massive concurrent development.

The need for *rapid response to change* exacerbates these risks and challenges. Traditional requirements management and change-control processes are no match for the large volumes of change-traffic across the multitudes of suppliers and operational stakeholders involved in a SISOS. Furthermore, many of the sources of change (externally interoperating systems, COTS products) are outside the program's span of control.

The criticality, software-intensiveness, and cross-cutting nature of many of these changes mean that traditional project organizations with software-element managers buried deep in the management structure will not meet the challenge of rapid and effective response to change. And tradi-

tional contracting mechanisms and incentive structures optimized around low-cost delivery to a fixed specification will have exactly the wrong effect on rapid cross-supplier adaptation to change. Technically, software architectures sacrificing ease of change for incremental computer system performance gains will also make rapid change unachievable, and much more up-front work on architecture trade-off analysis is needed to get the right balance among performance, dependability, ease of use, and adaptation to change.

The benefits of *free upgrades to COTS software* made to adapt to change also have risks and challenges. COTS changes are determined by COTS vendors. Each time this happens, the SISOS integrators are presented with a difficult challenge over which they have limited control [7]. And on a SISOS, it will happen a lot. Four years of survey data from the annual U.S. Air Force (USAF)/Aerospace Corporation Ground Systems Architectures Workshops indicate that the average COTS product in the satellite ground systems domain undergoes a new release every eight to 10 months. On a complex SISOS with dozens of suppliers making commitments to more than 100 different COTS products, at least 10 new releases will impact the program every month. Also, vendors typically support only the three most recent releases.

The Total System Performance Responsibility (TSPR) acquisition structure is not viable for a SISOS. This is due to management's need to coordinate supplier commitments to potentially incompatible COTS and nondevelopmental item (NDI) software components. Leaving dozens of suppliers of component systems with the TSPR authority to make hundreds of commitments to incompatible COTS and NDI components will not work. However, centralized management of most supplier decisions will not work either. Also, there is a significant risk of supplier micromanagement if the SISOS system integrator is a contractor staffed by people more familiar with making detailed development decisions versus making acquisition leadership decisions. There is a need to balance the acquisition strategy to determine *how much commonality is enough* for each aspect of the SISOS.

Lastly, the software benefits of enabling *early fielding of partial capabilities* come with risks of over-optimizing on the early capabilities and over-optimistically assuming that any sort of software architecture and code can be easily modified later. This assumption is invalid for most software; empirical data shows that the cost of software changes on large projects goes up by a factor of about 100 from requirements specification to post-deployment change [8]. This factor can be reduced significantly by thorough software architecting for change and risk management, as on the USAF/TRW Command and Control Processing Display System-R Project [9].

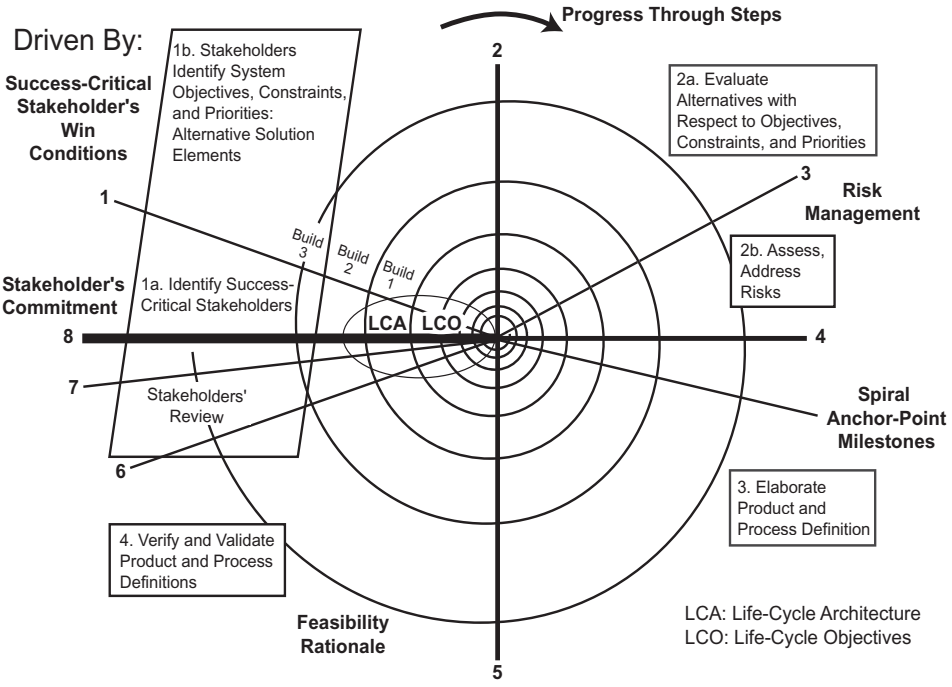

Figure 1: *The Win-Win Spiral Model*

Win-Win Spiral Model Overview

Current DoD acquisition policy in DoD Directive 5000.1 and DoD Instruction 5000.2 strongly emphasizes using evolutionary acquisition and spiral development [5, 6]. Figure 1 summarizes the Win-Win Spiral Model used on probably the largest and most transformational system of systems under development today: the U.S. Army/Defense Advanced Research Projects Agency Future Combat Systems Program. The model includes the following highlighted strategy elements:

- **Success-critical stakeholders' win conditions.** All of the project's success-critical stakeholders participate in integrated product teams (IPTs) or their equivalent to understand each other's needs and to negotiate mutually satisfactory (win-win) solution approaches.
- **Risk management.** The relative risk exposure of candidate solutions and the need to resolve risks early drives the content of the spiral cycles. Early architecting spirals likely will be more analysis-intensive; later incremental or evolutionary development spirals will be more code-intensive. However, all spirals can and should be concurrently engineering their analysis products and code.
- **Spiral anchor-point milestones.** These focus review objectives and commitments to proceed on the mutual compatibility and feasibility of concurrently engineered artifacts (plans, requirements, design, and code) rather than on individual sequential artifacts.
- **Feasibility rationale.** In anchor-point milestone reviews, the developers provide a feasibility rationale detailing evidence obtained from prototypes, models, simulations, analysis, or production code that supports a system built to the specified architecture and does the following:
  - Support the operational concept.
  - Satisfy the requirements.
  - Be faithful to the prototype(s).
  - Be buildable within the budgets and schedules in the plan.
  - Have all major risks resolved or covered by a risk-management plan.
  - Have its key stakeholders committed to support the full life cycle.

Having inadequate evidence is grounds for failing the review, unless shortfalls in the evidence are identified as risks and covered by satisfactory risk-management plans. Progress toward achieving a feasibility rationale for the project's artifacts is a much better progress indicator than percent-completeness of requirements or design specifications.

Further description of the Win-Win Spiral Model is in [3, 4], and detailed guide-

lines on its use are at <http://sunset.usc.edu/research/MBASE> [10].

Top 10 SISOS Risks and Spiral Mitigation Strategies

Here is a prioritized top-10 list of SISOS risks based on our SISOS experiences in Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance systems; space systems; the Army Future Combat Systems program; the U.S. National Air Traffic Control System; and commercial network-centric systems of systems, along with the results of a number of DoD Tri-Service Assessment Initiative reviews [11].

Risk 1: Acquisition Management and Staffing

The *biggest risk* in acquiring a SISOS is committing to acquisition practices and strategies that may still work for some systems but are incompatible for a SISOS. Often this occurs via legacy policies and cultures that assume SISOS requirements can be predetermined and allocated to hardware, software, and humans before architecting the SISOS and contracting for its component systems. For example, the Software Engineering Institute's Capability Maturity Model® Requirements Management Key Process Area says, "Analysis and allocation of the system requirements is not the responsibility of the software engineering group but is a prerequisite for their work" [12].

Actually, many SISOS requirements emerge with development and use rather than being pre-specifiable. Feasible technical requirements emerge through development and prototyping experience; feasible human-computer interface and decision support requirements emerge through software and system exercise and use.

The Win-Win Spiral Model addresses this risk through its risk-driven concurrent engineering and evolutionary development of SISOS products and processes. Mature, highly precedented systems mostly can be pre-specified with low risk; immature systems or unprecedented combinations of systems may need several spiral cycles of risk resolution to get the right combination of requirements, architecture, system elements, and life-cycle plans.

The *second major risk* is the lack of rapid response to change that happens in traditional project organizations where software expertise and decision authority are scattered at low management levels across various project elements. Instead, a project needs an integrated software and information processing leader reporting directly to the project manager, with strong management through the counterpart software and information processing leaders who report directly to each IPT leader and system-supplier project manager.

The project also needs strong software networking within the SISOS IPTs, which may include IPTs for sensors, networks and communications, command and control, ground/sea/air/space vehicles, logistics, training, integration and test, modeling and simulation, and infrastructure. Further, it needs collaborative supplier integration and support of concurrent incremental development as discussed in Risk 4 and Risk 5.

The *third major risk* is key staff shortages and burnout. The key system and software personnel on a SISOS have little time to do their assigned work after participating in all or most of the coordination meetings that a SISOS requires. Further, a SISOS evolutionary acquisition project can go on for years, leading to a high risk of staff burnout.

Risk mitigation practices include career path development, mentoring junior staff to provide replacements for key personnel, incremental completion bonuses, flowdown of contract award fees to project performers, and recognition initiatives for valued contributions.

Risk 2: Requirements/Architecture Feasibility

The biggest risk here is committing to a set of requirements or architecture without validating feasibility. Requirements/architecture nature and criticality were exemplified by the premature commitment to a one-second-response time requirement by a project discussed in [3]. The project had to throw away 15 months' work in architecting a custom $100 million system to meet the one-second requirement when a prototype belatedly showed that a $30 million COTS-based system with a four-second-response time would be sufficient.

Generally, requirements/architecture infeasibilities regarding quality factors such as response time, throughput, security, safety, interoperability, usability, or evolvability have the highest risk exposures. They are discussed further in Risk 6. The Win-Win Spiral Model's anchor-point milestone pass/fail criteria and feasibility rationale explicitly address this risk. They prevent a project's marrying its architecture in haste and having to repent at leisure – that is, if any leisure time is available.

Risk 3: Achievable Software Schedules

In the past, software cost has been the most critical resource constraint. The large volume of software in a SISOS tends to put the software development schedule on the project's critical path more so than for simple systems. Table 1 clearly shows the magnitude of this risk.

The Win-Win Spiral Model's anchor-point milestones and feasibility rationale again directly address this issue. Schedule feasibility should be addressed both by software cost and schedule estimation models (using and comparing the results of two independent models is a good practice), and by explicit development and critical-path analysis of project activity networks (probabilistic activity networks are more conservative and realistic). The software development time shown in Table 1 can be reduced by the major techniques for increasing overall software productivity (software reuse, COTS, reducing rework, top personnel, and better tools), plus the following two techniques that focus on improving schedule directly.

Architecting and Organizing for Massive Concurrent Development

If the SISOS could be architected so that supplier-developed components could instantly plug and play, Table 1 indicates that organizing the project into many 300-KSLOC components would get the job finished in 33 months versus 108 months. Unfortunately, the effects of architectural imperfection and continuing change make seamless integration an unrealistic objective, but the relative gains of reducing integration rework are worth trying to achieve.

The other main problem is the fraction of development time it takes to produce a fully validated integration architecture, which has been shown to increase with the amount of software needing to be integrated. Figure 2 [13] shows how this tradeoff between architecting time before finalizing SISOS supplier specifications and rework time can be analyzed by the Constructive Cost Model (COCOMO) II Architecture and Risk Resolution Factor [14]. It shows that for a 10,000 KSLOC SISOS, the sweet spot minimizing the sum of both architecting and rework time occurs at about 37 percent of the development time, with a relatively flat region between 30 percent and 50 percent. Below 30 percent, the penalty curve for premature issuance of supplier specifications is steep: A 17 percent investment in architecting yields a rework penalty of 48 percent for a total delay of 65 percent compared with a rework penalty of 20 percent and a total delay of 50 percent for the 30 percent architecting investment.

This curve and its implications were

® Capability Maturity Model is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

convincing enough to help one recent SISOS add 18 months to its schedule to improve architectural specifications.

The Schedule as Independent Variable Process

The schedule as independent variable (SAIV) process [15] is a special case of the Win-Win Spiral Model that applies when there is a strong need to produce an initial operational capability (IOC) by a particular date, but the exact nature of the IOC is not well specifiable in advance. The SAIV process, which is compatible with the Win-Win, incremental, and concurrent development processes operates as follows:

- Work with stakeholders in advance to achieve a shared product vision, realistic expectations, and prioritized requirements.
- Estimate the maximum size of software buildable with high confidence within the available schedule.
- Define a core-capability IOC content based on priorities, end-to-end usability, and need for early development of central high-risk software.
- Architect the system for ease of dropping or adding borderline-priority features.
- Monitor progress; add or drop features to accommodate high-priority changes or to meet schedule.

The SAIV process has been used successfully to date on all sizes of SISOS. For example, lower-priority requirements originally within one SISOS program's IOC set such as automatic real-time natural language translation were deferred to create an achievable core-capability IOC.

Risk 4: Supplier Integration

As the SISOS system suppliers integrate their architectures and components and jointly respond to changes, they will need to share information and rapidly collaborate to negotiate changes in their products, interfaces, and schedules. The COCOMO II *team cohesion* factor yields an added 66 percent in effort and up to 30 percent in added schedule between seamless team cohesion and very low team cohesion.

In mitigating these risks, the win-win aspects of the Win-Win Spiral Model became paramount. Strategies for achieving win-win supplier participation include making them first-class stakeholders in negotiating their parts of SISOS objectives, constraints, priorities, and preferred alternatives in Figure 1; establishing early training and team-building activities for selected suppliers; proactively identifying needs for supplier collaboration and networking of their lead software and system architects; and establishing contract provisions and award fee criteria for effective collaboration in such areas as schedule preservation, continuous integration support, cost containment, technical performance, architecture and COTS compatibility, and program management and risk management. An example award fee evaluation process and criteria are provided in [16]. One recent large SISOS program has implemented a similar *shared destiny* process into its supplier contracting.

Risk 5: Adaptation to Rapid Change

As discussed earlier, adaptation to change is a SISOS necessity, but continuous adaptation to change across dozens of suppliers, IPTs, and external interoperators can be completely destabilizing. Within the Win-Win Spiral Model, the best strategy for balancing change and stability is incremental development. As seen in Figure 1, the spiral cycles combine architecting and development with key parts of high-risk elements developed in a Build 1 and used as part of the feasibility rationale for the SISOS Life-Cycle Architecture (LCA) milestone. The post-LCA builds have the suppliers concurrently developing increments of capability within the validated architecture established at the LCA milestone.

To stabilize development, proposed changes are deferred as much as possible to later builds, and the SAIV process can be used to drop lower-priority features not needed by other suppliers to keep on a common schedule. This process is similar to the Microsoft synchronize-and-stabilize process [17] and works best if there is some slack built into the end of each build. Other strategies for adaptation to rapid change include proactive technology-watch, COTS-watch, and interoperability-watch activities; cross-supplier and cross-IPT networking; change-anticipatory architectures; and agile change control and version control capabilities.

An example of the success of these practices has been the Internet Spiral Process [18] used to adapt and evolve the Internet well before the formalization of the spiral model.

Risk 6: Software Quality Factor Achievability

As discussed in Risk 2, software quality factors are the most difficult sources of SISOS requirements/architecture feasibility risk. These factors are strongly scenario-dependent and interact in complex ways that cut across supplier and IPT boundaries. A good example is a vehicle self-defense timeline, which imposes performance requirements and trade-offs across sensor, networking, fusion, command-control, software infrastructure elements of a SISOS and more, along with additional trade-offs between performance, security, usability, safety, and fault tolerance.

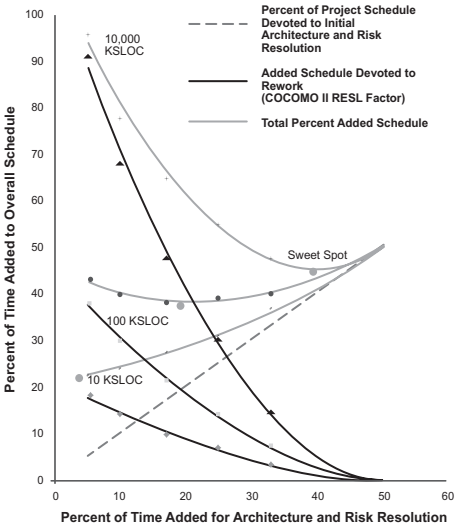

Figure 2: *How Much Architecting Is Enough?*

A key Win-Win Spiral strategy for quality factor achievability is to establish a quality factor trade space by replacing single-value quality factor requirements with a range between acceptable and desired values. This provides the system and software architects with enough degrees of freedom to converge to a mutually acceptable – or win-win – combination of achievable quality factor values. Another key strategy is using the SEI's Architecture Trade-off Analysis Method (ATAM) for stakeholder establishment, prioritization, and assessment of quality factor values achievable with a given architecture, and identification of strategies to bring the values up to acceptable levels. Several examples of successful ATAM use are provided in [19].

Risk 7: Product Integration and Electronic Upgrade

The SISOS software needs to be integrated across supplier hierarchies, IPT domains, computing platforms, vehicle platforms, critical scenarios, operational profiles, system modes, and friendly-to-adversarial environments. Having too much or too little concurrency across these dimensions of integration can cause significant delays and rework. This rework needs to be fed back to developers who will already be busy developing the next build, causing even further SISOS delays.

The benefits of electronic software upgrades discussed at the start of this article come with several types of version mismatch risks. Examples include putting the wrong version's upgrades onto a platform

in the field or having different fielded platforms running different versions of the SISOS software. These mismatches can cause software crashes, communication outages, out-of-synchronization data, or mistaken decisions.

Key Win-Win Spiral Development strategies for addressing these risks include up-front involvement of software-oriented integration, test, supportability, and maintenance stakeholders in win-win negotiations affecting stakeholders' ability to perform; early establishment, usage, and incremental growth of software and system integration laboratories for the overall SISOS and for its key IPT areas; and architecting the software to accommodate continuous operation and synchronized upgrades (for example, by enabling parallel operations of old and new versions while validating and synchronizing an upgrade). Again, the Internet is a highly successful example.

Risk 8: Software COTS and Reuse Feasibility

The first two sections in this article included discussion of the benefits of free COTS software changes and some of the SISOS risks involved in synchronizing COTS upgrades across a wide variety of independently evolving COTS products. For a SISOS with many suppliers developing ambitious capabilities within tight budgets and schedules of more than 30 months, the temptation is to not upgrade the COTS products and to deliver unsupported versions [20]. In one case, we encountered a large system delivered to the customer and users with 55 of its 120 COTS products operating on unsupported releases.

Win-Win Spiral Development mitigation strategies for COTS-related risks include contract provisions prohibiting the delivery of unsupported COTS components; establishing key COTS vendors as strategic partners and success-critical stakeholders; proactive COTS-watch experimentation and participation in user groups (for example, to cover security and real-time performance concerns), operating a SISOS-wide COTS product and version tracking and compatibility analysis activity; and developing and executing a strategy for periodic synchronized COTS upgrades.

Software reuse is a powerful strategy for reducing software cost and schedule, but frequently estimates of 80 percent software reuse on a suppliers' system turn out to be more like 40 percent once the different natures of the SISOS and the legacy software are recognized. Win-Win Spiral Development strategies for mitigating software reuse risks include validating the compatibility of supplier reuse components with SISOS product line architectures, constraints, and assumptions; continuous data analysis of actual versus estimated reuse parameters and recalibration of reuse estimates; and performing root cause analyses of reuse successes and failures. Further reuse and product line best practices and successful examples can be found in [21] and [22].

Risk 9: External Interoperability

Large SISOS are likely to require interoperability with more than 100 independently evolving external systems (and even more if COTS components are included). As with COTS, there are major risks of some or all of the SISOS systems getting out of synchronization with these external systems. Major Win-Win Spiral Development strategies for risk mitigation include establishing proactive stakeholder win-win relationships with critical interoperability systems, including memoranda of agreement on interoperability preservation; proactive participation in the evolution of the Joint Capabilities Integration and Development System [23]; operating an external systems interoperability tracking and compatibility analysis activity; and inclusion of external interoperability in modeling, simulation, integration, and test capabilities. Here again, the Internet provides an excellent example.

Risk 10: Technology Readiness

The scale and mission scope of a SISOS may far exceed the capabilities of technologies that have demonstrated considerable maturity in smaller systems or friendlier environments. Examples are adaptive mobile networks, autonomous agent coordination capabilities, sensor fusion capabilities, and software middleware services. Assuming that a technology's readiness level on a smaller system will be valid for a SISOS runs a major risk of performance shortfalls and rework delays.

Primary Win-Win Spiral Development risk mitigation strategies focus on satisfying a feasibility rationale for the key advanced technologies, including the exercise of models, simulations, prototypes, benchmarks, and working SISOS applications on representative SISOS normal, crisis, and adversarial scenarios. Risk management strategies include identifying fallback technology capabilities in case key new technologies prove inadequate for SISOS usage. These practices are all consistent with the guidance in DoD Instruction 5000.2.

Conclusion

Competitive pressures for increased integration and high performance of commercial, industrial, and public services capabilities such as military defense or homeland security are leading to multi-domain and multi-supplier systems of systems, which are increasingly software-intensive. Acquiring such a SISOS has many differences from acquiring traditional systems. Besides the significantly larger numbers of options, changes, suppliers, and domains to accommodate, there are significantly larger numbers of external interfaces, COTS products, coordination networks and meetings, operational stakeholders, and emergent versus pre-specifiable requirements.

These differences in scope, scale, and dynamism have made traditional acquisition practices increasingly inadequate. Current initiatives toward evolutionary acquisition and spiral development are promising but many new practices need to be worked out. Experiences on several SISOS have identified both a set of top-10 SISOS risks and corresponding risk mitigation strategies currently being applied on some SISOS.

Applying the corresponding risk mitigation strategies within a Win-Win Spiral Development and evolutionary acquisition process is meeting with some success, and appears to be a good starting point for identifying and coping with SISOS risks. But much more experience on SISOS acquisition and development will be needed to achieve mature SISOS acquisition capabilities.◆

References

1. Harned, D., and J. Lundquist. "What Transformation Means for the Defense Industry." The McKinsey Quarterly 3 Nov. 2003: 57-63.
2. Rechtin, E., and M. Maier. The Art of Systems Architecting. 2nd ed. CRC Press, 2001.
3. Boehm, B., and W. Hansen. "The Spiral Model as a Tool for Evolutionary Acquisition." CrossTalk May 2001: 4-11.
4. Boehm, B., and D. Port. "Balancing Discipline and Flexibility With the Spiral Model and MBASE." CrossTalk Dec. 2001: 23-28.
5. DoD Directive 5000.1. "The Defense Acquisition System." Washington, D.C.: U.S. Department of Defense, 12 May 2003.
6. DoD Directive 5000.2. "Operation of the Defense Acquisition System." Washington, D.C.: U.S. Department of Defense, 12 May 2003.
7. Meyers, B.C., and P. Oberndorf. Managing Software Acquisition: Open

Systems and COTS Products Addison-Wesley, 2001.
8. Boehm, B., and V. Basili. "Software Defect Reduction Top-10 List." Computer Jan. 2001: 135-137.
9. Royce, W.E. Software Project Management. Addison-Wesley, 1998.
10. USC-Center for Software Engineering. "Guidelines for Model-Based (System) Architecting and Software Engineering." Los Angeles: University of Southern California, 2003.
11. McGarry, J., and R. Charette. "Systemic Analysis of Assessment Results from DoD Software-Intensive System Acquisitions." Tri-Service Assessment Initiative Report. Washington, D.C.: OSD Defense, Acquisition, Technology, and Logistics, 2003.
12. Paulk, M., et. al. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley, 1994.
13. Boehm, B., and R. Turner. Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley, 2004.
14. Boehm, B., et al. Software Cost Estimation With COCOMO II. Prentice Hall, 2000.
15. Boehm, B., et al. "Using the Spiral Model and MBASE to Generate New Acquisition Process Models: SAIV, CAIV, and SCQAIV." CROSSTALK Jan. 2002: 20-25.
16. Reifer, D., and B. Boehm. "A Model Contract/Subcontract Award Fee Plan for Large, Change-Intensive Software Acquisitions." Los Angeles: USC Center for Software Engineering, Apr. 2003.
17. Cusumano, M., and R. Selby. Microsoft Secrets. The Free Press, 1995.
18. U.S. Air Force Scientific Advisory Board. "Information Architectures That Enhance Operational Capability in Peacetime and Warfare." Washington, D.C.: U.S. Air Force, Feb. 1994.
19. Clements, P., R. Kazman, and M. Klein. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley, 2002.
20. Basili, V., and B. Boehm. "COTS-Based Systems Top-10 List." Computer May 2001: 91-93.
21. Reifer, D. Practical Software Reuse. John Wiley and Sons, 1997.
22. Clements, P., and L. Northrop. Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.
23. Joint Chiefs of Staff Manual. "Operation of the Joint Capabilities Development System." CJCSM 3170.01. Washington, D.C.: Chairman of the Joint Chiefs of Staff, 24 June 2003.

About the Authors

Barry Boehm, Ph.D., is the TRW professor of software engineering and director of the Center for Software Engineering at the University of Southern California. He was previously in technical and management positions at General Dynamics, Rand Corp., TRW, and the Office of the Secretary of Defense as the director of Defense Research and Engineering Software and Computer Technology Office. Boehm originated the spiral model, the Constructive Cost Model, and the stakeholder win-win approach to software management and requirements negotiation.

University of Southern California
Center for Software Engineering
Los Angeles, CA 90089-0781
Phone: (213) 740-8163
(213) 740-5703
Fax: (213) 740-4927
E-mail: boehm@sunset.usc.edu

A. Winsor Brown is a Senior Research Scientist and Assistant Director of the University of Southern California Center for Software Engineering. As an engineer with decades of experience in large and small commercial and government contracting companies, he started his career in computer hardware design but shifted to software within months and remains there today. He has a Bachelor of Engineering Science from Rensselaer Polytechnic Institute and a Masters of Science in Electrical Engineering from California Institute of Technology.

University of Southern California
Center for Software Engineering
941 West 37th Place
Los Angeles, CA 90089-0781
Phone: (714) 891-6043
Fax: (213) 740-4927
E-mail: awbrown@cse.usc.edu

Victor R. Basili, Ph.D., is Professor of Computer Science at the University of Maryland, College Park and the Executive Director of the Fraunhofer Center – Maryland. He was a founder of the Software Engineering Laboratory at NASA Goddard Space Flight Center. He works on measuring, evaluating, and improving software processes and products. Basili has received several awards, including the 2000 Association for Computing Machinery (ACM) SIGSOFT Outstanding Research Award and the 2003 Institute of Electrical and Electronics Engineers (IEEE) Computer Society Harlan Mills Award. He is an IEEE and ACM Fellow.

Fraunhofer USA Center for
Experimental Software Engineering
University of Maryland
4321 Hartwick RD
STE 500
College Park, MD 20742-3290
Phone: (301) 403-8976
Fax: (301) 403-8976
E-mail: basili@cs.umd.edu

Richard Turner, D.Sc., is a member of the Engineering Management and Systems Engineering Faculty at The George Washington University in Washington, D.C. Currently, he is the assistant deputy director for Software Engineering and Acquisition in the Software Intensive Systems Office of the Under Secretary of Defense (Acquisition, Technology, and Logistics). Turner is co-author of the book "CMMI Distilled."

1931 Jefferson Davis HWY
STE 104
Arlington, VA 22202
Phone: (703) 602-0581 ext. 124
E-mail: rich.turner.ctr@osd.mil