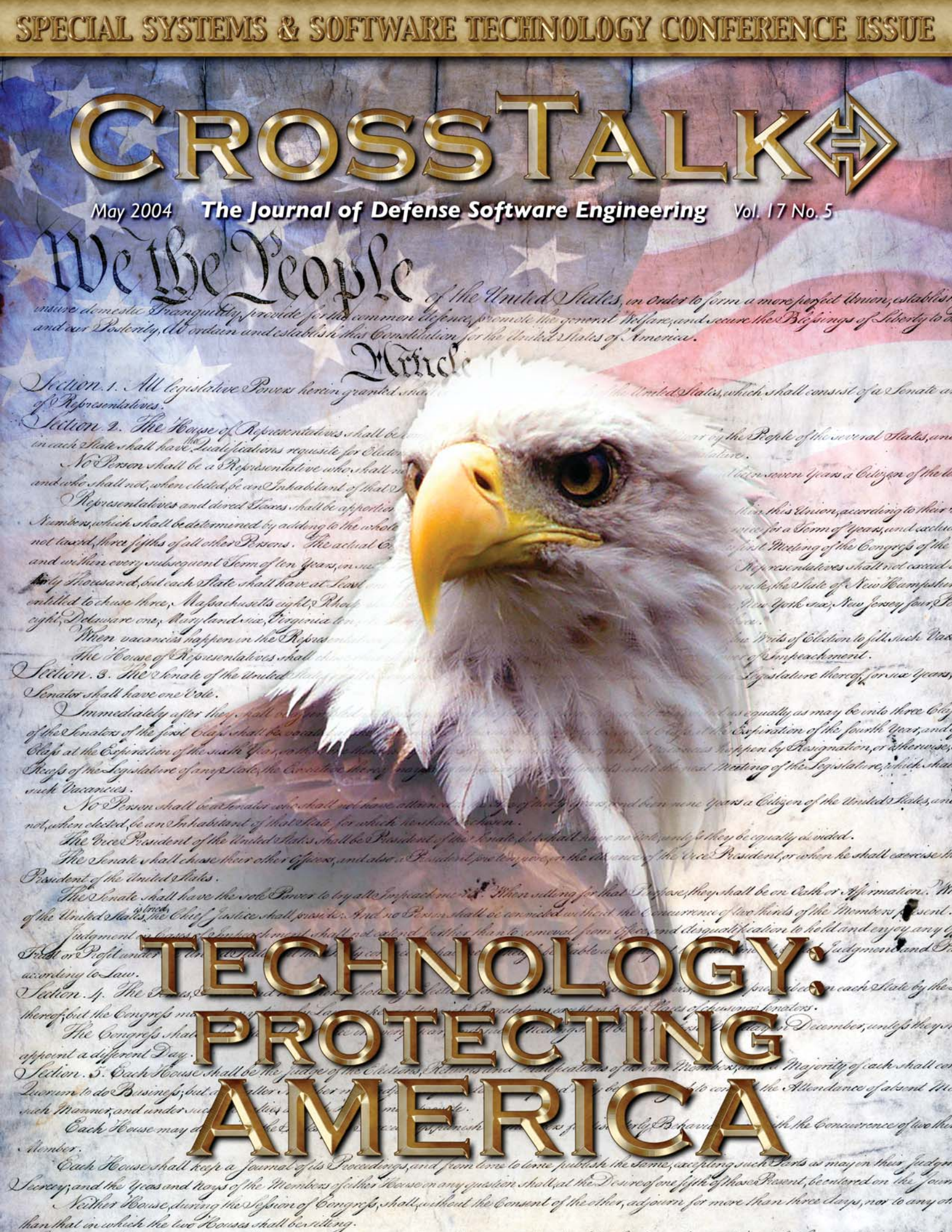
SPECIAL SYSTEMS & SOFTWARE TECHNOLOGY CONFERENCE ISSUE

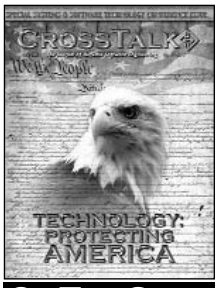
CrossTalk

May 2004 The Journal of Defense Software Engineering Vol. 17 No. 5



TECHNOLOGY: PROTECTING AMERICA

Best Practices



ON THE COVER
Cover Design by Kent Bingham.

Software Engineering Technology

Open Forum

Departments

Note: For an update on CROSSTALK's future status, please visit our Web site at <www.stsc.hill.af.mil>.

CROSSTALK

PUBLISHER *Tracy Stauder*
ASSOCIATE PUBLISHER *Elizabeth Starrett*
MANAGING EDITOR *Pamela Palmer*
ASSOCIATE EDITOR *Chelene Fortier-Lozancich*
ARTICLE COORDINATOR *Nicole Kentta*
CREATIVE SERVICES COORDINATOR *Janna Kay Jensen*
PHONE (801) 586-0095
FAX (801) 777-8069
E-MAIL crosstalk.staff@hill.af.mil
CROSSTALK ONLINE www.stsc.hill.af.mil/crosstalk

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 25

Ogden ALC/MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CROSSTALK.

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC Online Services: www.stsc.hill.af.mil
Call (801) 777-7026, e-mail: randy.schreifels@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CROSSTALK available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

Technology By Any Other Name

I love technology. Whether the technology is a new product, a new development method, a new process, or even a new tool, I enjoy learning about, studying, and using it. The word technology itself seems so encompassing. It is the creative element for engineers and scientists, the power behind programming, and the glue that bonds the triad of people, process, and product. Most importantly, technology allows adults to connect with their inner child and play with a new toy. Since I'm a part of the Software Technology Support Center, I believe I have a responsibility to stay abreast of the latest hardware, software, etc. However, I'm often accused of playing with toys. My response is invariably, "I'm just doing my job."

At home, my quest to work with new technology has allowed me to become skilled at home repair. I gladly assist in fixing anything broken, but one must have the proper tools to do the proper job. Searching for the proper tools can be a time-consuming process, which I have found from actual experience can consume the better part of a Saturday morning. I try to explain to my wife that selecting the proper tools is part of project planning. I think she believes me.

In this month's CROSSTALK, we spotlight some new technologies by featuring some of the presentations from the 2004 Systems and Software Technology Conference (SSTC) held April 19-22 in Salt Lake City. The conference allowed thousands of professionals a chance to learn about new technologies related to building systems and software, and gave them some new tools to use. If you missed SSTC, this issue provides some highlights.

If you develop systems, don't miss the discussion of the Win-Win Spiral Development Model by Dr. Barry Boehm, A. Winsor Brown, Dr. Victor Basili, and Dr. Richard Turner in *Spiral Acquisition of Software-Intensive Systems of Systems*. Next, Gregory S. Shelton, Randy Case, Louis P. DiPalma, and Dan Nash share information regarding the ontology of various architectures in *Advanced Software Technologies for Protecting America*. In *Bridging Agile and Traditional Development Methods: A Project Management Perspective* by Paul E. McMahon, you will learn about strategies for managing a project based on agile development methods. Next, Udai K. Kudikyala and Dr. Rayford B. Vaughn Jr. explain some of the issues that relate to pathfinder networks in *Understanding Software Requirements Using Pathfinder Networks*. Dr. Richard Bechtold details some of the pitfalls of testing commercial off-the-shelf products in *Efficient and Effective Testing of Multiple COTS-Intensive Systems*. Also, don't miss Theron R. Leishman's and Dr. David A. Cook's discussion of managing risks in *Risk Factor: Confronting the Risks That Impact Software Project Success*.

I hope this issue of CROSSTALK sparks an idea or two that will be useful to you in your work. It may not be as fun as a trip to Home Depot, but it won't require an entire Saturday morning either. As for me, I'm off to explore more technology.

U.S. Government's Top 5 Quality Software Projects

CROSSTALK is proud to announce the following winners of the 2003 U.S. Government's Top 5 Quality Software Projects. Thank you to everyone who submitted nominations. This year's awards were presented at the 2004 Systems and Software Technology Conference.

- **Advanced Field Artillery Tactical Data System**
  Customer: U.S. Army/USMC/ U.S. Navy
  Developer: PM Intelligence and Effects and Raytheon Team

- **Defense Medical Logistics Standard Support**
  Customer: Military Health System
  Developer: DMLSS Program Office

- **H1E System Configuration Set**
  Customer: Program Manager Air, AIR-265
  Developer: F/A-18 Advanced Weapons Laboratory and Boeing IDS

- **OneSAF Objective System**
  Customer: Program Manager OneSAF Objective System – U.S. Army's Program Executive Office for Simulation, Training, and Instrumentation
  Developer: OneSAF Objective System Integrated Product Team

- **Patriot Excalibur**
  Customer: AFMC
  Developer: 46 TW/XPI (TYBRIN)

Brent D. Baxter
SSTC Technical Conference Manager
Manager, Software Technology Support Center

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].

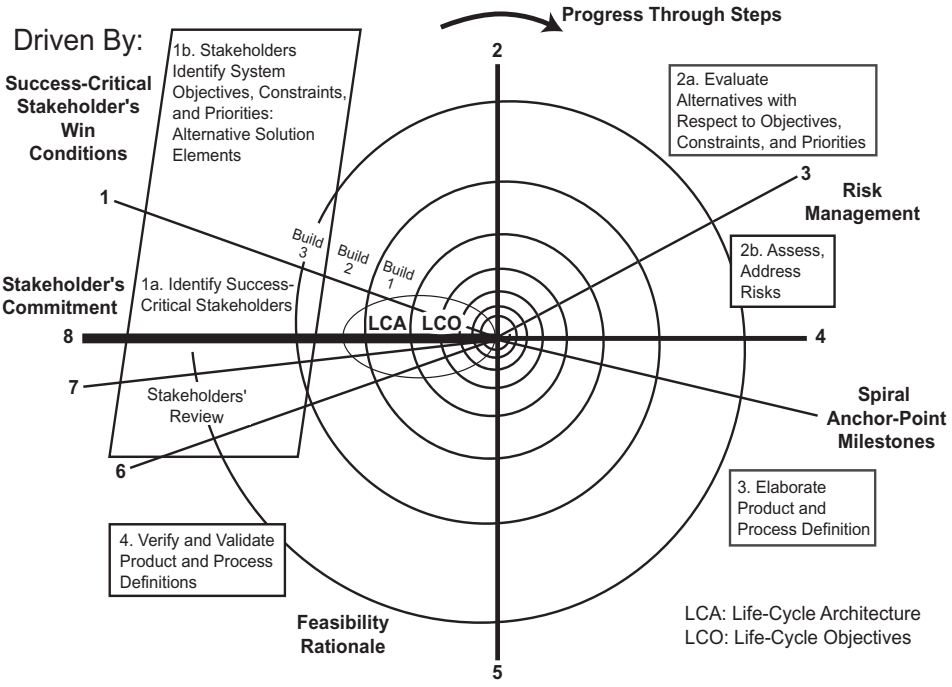


Figure 1: *The Win-Win Spiral Model*

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.

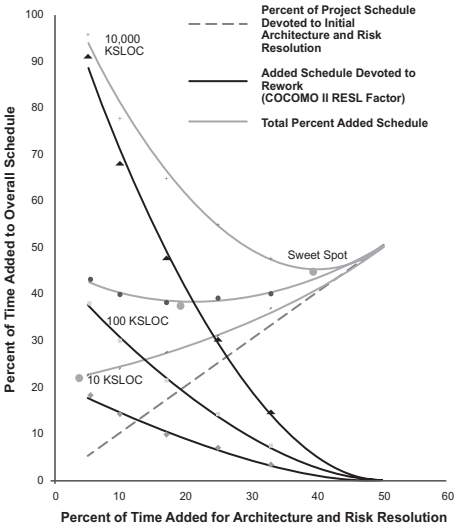


Figure 2: *How Much Architecting Is Enough?*

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

Advanced Software Technologies for Protecting America

Gregory S. Shelton, Randy Case, Louis P. DiPalma, and Dan Nash
Raytheon Company

Thursday, 22 April 2004
Speaker Luncheon: 12:20 - 1:20
Marriott Grand Ballroom

Advanced software technologies are required for the success of homeland security, missile defense, intelligence/surveillance/reconnaissance, and precision engagement. State-of-the-art software technologies for system architecture development such as model-driven computing, reference architectures, and supporting technology enablers are needed for these critical systems.

The events of 9-11 in America and the ongoing actions throughout the world have keenly focused our thoughts on issues of protection and homeland security. Portions of the solutions to these problems will be in better human intelligence, greater diligence, and resources applied to traditional security. However, technology-driven solutions are needed to better increase the use of in-place resources and meet newer threats.

Four system areas are vital to protecting America: homeland security, missile defense, intelligence/surveillance/reconnaissance (ISR), and precision engagement. These all require advanced software technologies that will enable the development of integrated mission systems. These technologies go beyond existing software technologies traditionally focused on *stovepipe* software component or platform solutions. Technologies supporting system architecture development are important for mission success.

Homeland Security

The threat of terrorist attacks in the United States brings into vivid focus the need to harness technology to detect threats and protect against and respond to them. Table 1 presents a list of some recent initiatives directly related to homeland security; the applicable enabling advanced software technologies are also listed. In some *stand-alone* activities such as bomb detection or airline missile protection, no new software technologies are needed. More work in domain-specific algorithms may be required, but fundamental software techniques are adequate for these programs to succeed.

Common to many homeland security programs is the need for searching, mining, and analyzing large databases (for example, visa tracking, biometric pattern matching, and analysis of foreign language materials). The fundamentals of these types of database technologies exist and upgrades in technologies are ongoing, particularly in enhancements to speed and accuracy.

New needs to integrate communication systems from agencies that formerly did not use common equipment (police, fire, etc.) and the need to fuse information such as weather data and models of chemical/biological agents requires the integration of existing system architectures. Tools and techniques to develop these software-intensive system architectures such as using ontology for information definition/retrieval and using reference architectures are needed for the successful development of these systems.

Table 1: *Software Technologies Needed for Homeland Security*

Detection Systems	Advanced Software Technologies Needed
Airport Bomb Detection	• Software Technologies Are Adequate
Open Source Analysis	• Search Engines • Automatic Language Translation • Data Mining
Entry/Exit Visa Tracking	• Data Mining • Predictive Analysis
Protection Systems	
Biometrics	• Intelligent Database Searching
Commercial Airline Missile Protection	• Software Technologies Are Adequate
Response Systems	
Integrated Communications Systems (Fire, Police, National Guard, etc.)	• Information Organization/Retrieval Using Ontology • Context-Sensitive Reference Architectures
Chemical/Biological Agent Response	• Data Fusion for Virtual Weather Modeling

Missile Defense

Recent developments in world events and national policy have renewed the dialogue on missile defense. The mission of missile defense is to defend successfully against missiles of all ranges (short, intermediate, and long) in all phases of flight (boost, midcourse, and terminal). All components must be fully networked to assure coordinated operations with very short timelines. An operational missile defense system must have three fundamental technical capabilities and associated software technologies: sensors, interceptors, and battle management, command, and control (BMC2), as shown in Table 2.

The sensor components (radar, infrared, and electro-optical) have been developed and will continue to be matured. We are seeing model-based software techniques used to support the definition of architectures and generation of executable code for some of these applications. The interceptor components of these systems require software data fusion approaches and system architectures to better enable the data fusion. The most software-intensive portion of missile defense is the BMC2 component. The need for handling large volumes of information accurately and within very short timelines places demands on the development of effective system architectures. This area requires a host of advanced software techniques to develop effective system architectures, as used in software techniques to aid human decision making (intelligent agents, cognitive computing techniques, etc.).

ISR

The ISR programs cover the full spectrum of information management, providing the ability to task, collect, process, exploit, and disseminate national and tactical target data (see Table 3). These abilities are crucial for warfighters to achieve information dominance throughout the entire battlespace. The ISR activities are typically composed of tasking, collection, and activities related to processing/exploitation/dissemination.

A key attribute of ISR is the system integration of multiple sensors, platforms, and networks. This *system of systems* is characterized by the need for well-defined system architectures to support the needed interoperability and integration. New software technologies common to all tasks in ISR include ontology for information management, reference architectures, and model-driven computing architectures. Advances in data mining and intelligent agents will expedite handling of large information volumes in real time. Interoperability and information dissemination to various users will require new techniques to handle multi-level security issues.

Precision Engagement

Precision engagement systems enhance America's defense by providing warfighters with highly accurate, adverse weather, rapid sensor-to-shooter capabilities required on today's battlefields (see Table 4, next page). Precision engagement works in conjunction with ISR to provide a wide range of capabilities.

The information from ISR that is needed to provide targeting for precision munitions requires using software techniques that support the development of system architectures (ontology, reference architectures, and model-driven architecture development). In particular, shorter sensor-to-shooter timelines require a system architecture construction optimized for time sensitivity.

Missile Defense Component	Advanced Software Technologies Needed
Sensors – Detect, acquire, and track target missiles; predict their path; identify a threat among decoys; and direct the interceptor to destroy the missile.	• Context-Sensitive Software Reference Architectures • Model-Driven Software Architectures
Interceptors – Seek, discriminate, and destroy targets.	• Data Fusion
BMC2 – Provides the commander with threat and tracking data from sensors, suggests the most effective response, directs interceptors to the target, and measures damage and effectiveness.	• Context-Sensitive Software Reference Architectures • Intelligent Software Agents • Human Factors Interactions With Complex Software Systems • Model-Driven Software Architectures • Cognitive Computing Techniques

Table 2: *Software Technologies for Missile Defense*

ISR Activity	Advanced Software Technologies Needed
Example ISR Tasking Systems • UAV Tactical Control System • Global Hawk Mission Control Element • Intelligence Satellites Control Element • Space-Based Infrared Systems (SBIRS) Control Element	• Information Organization/Retrieval Using Ontology • Context-Sensitive Reference Architectures • Intelligent Software Agents • Model-Driven Computing • Human Factors Interactions With Complex Systems
Example ISR Collection Systems • Global Hawk Integrated Sensor Suite • U-2 Advanced Synthetic Aperture Radar • Multi-Platform Radar Technology Insertion Program (MP-RTIP) • Rivet Joint Aircraft Sensors	• Information Organization/Retrieval Using Ontology • Context-Sensitive Reference Architectures • Intelligent Software Agents • Model-Driven Computing • Human Factors Interactions With Complex Systems • Data Mining • Multi-Level Security
Example ISR Process/Exploit/ Disseminate Systems • Cooperative Engagement Capability (CEC) • Global Broadcast Service (GBS) • National Polar-Orbiting Operational • Environmental Satellite System (NPOESS)	• Information Organization/Retrieval Using Ontology • Context-Sensitive Reference Architectures • Intelligent Software Agents • Model-Driven Computing • Human Factors Interactions With Complex Systems • Data Mining • Cognitive Computing Techniques • Multi-Level Security

Table 3: *ISR Systems Software Technologies*

Software Technologies for System Architecture Development – A Common Theme

Systems being deployed and developed for protecting America require advanced software technologies. In some cases, where the particular system architecture is standalone or composed of mostly point-to-point connections and limited broadcasting, the software approaches of today are sufficient. There will still be needed development of more capable algorithms and processors to support those algorithms, but the underlying software tools, paradigms, and enablers do not require further extensive research and development to be successful.

In many of the other above cases, we find as a common theme the need for existing software capabilities to be extended so that large-scale systems/platforms can work together to achieve the required missions. We believe that success in the new *system-of-systems* environment is enhanced by using software that will be more *intelligent* and developed as a direct offspring of modeling and simulation activities within the context of executable enterprise reference architectures. These technologies are being developed today at Raytheon, other defense contractors, and university/research organizations.

The left column of Table 5 (see next page) shows mature deployed software technologies used in defense applications today. The right column summarizes the software advances needed for the system types previously described. While these

technologies are in various states of maturity (including some such as data mining, which are fairly robust), they have not been widely deployed in key systems. Technologies for the development of system architectures are common to many of the systems needed for protecting America.

Looking at the key areas for defending and protecting America, we find that support for development of large, integrated mission systems is needed. The need for well-defined context-sensitive architectures is paramount for achieving these systems of systems such as Common Operating Picture (COP), DDX Destroyer, Future Combat System, or Joint Strike Fighter. The semantics of these large amounts of information are captured using ontological tools. The reference architectures are defined within the contexts of architecture frameworks. Finally, the architectures themselves are actually executable models supported by model-based, architecture-driven software development. Other enabling technologies such as cognitive computing and intelligent agents are all focused toward the software system development. Figure 1 illustrates the relationships of several key software technologies that will help realize the system architectures needed in the future.

Information Organization/ Retrieval Using Ontology

The initial step in developing large-scale system architectures is managing large-scale information semantics. Military knowledge workers are immersed in data smog. We have far more capability to create information than to find and retrieve relevant information. The result is huge amounts of amorphous, unstructured data that overwhelm us when we need pertinent, actionable data for informed decisions.

Technologies to help manage, search, and retrieve data include metadata for data descriptions, taxonomies for data categories, and ontology for data relationships (see Figure 2). Applications have been driven by commercial needs to identify information on the semantic Web and to provide Web services that deliver the right information to consumers. The value of such technologies to military applications is recognized by the Defense Advanced Research Projects Agency (DARPA), who sponsored development and deployment of a machine-processable ontology description language called the DARPA Agent Markup Language (DAML)[1].

Military information users must make life-critical decisions based on large amounts of time-sensitive, rapidly changing inputs from multiple sensors and sources. Having a single, consistently applied meaning for concepts, categories, and relationships reduces confusion, misinterpretation, and mistakes. Cognitive overload is reduced by supplying users with information that is relevant to their location, situation, and responsibilities. Ontology can be used to support both improvements. An example of where this applies is the Common Operating Picture (COP), which is a distributed database. Currently it is packed with disparate and incompatible data. In the future, human operators and software agents marking up information from sensors or sources in accordance with military standardization will generate it.

The Common Relevant Operating Picture is obtained by consumers (humans or software agents) subscribing to relevant information specified in accordance with the same ontology used in the creation of the COP.

Context-Sensitive Reference Architectures

Reference architectures (see Figure 3) bridge the gap between processes addressing the development of contingency operations for future systems and the implementation of domain-specific architectures that build on legacy systems while incorporating new technologies and capabilities. Modeling and simulation is a key tool to support evaluating the effectiveness of the reference architectures and the resulting domain-specific architectures.

The results of modeling and simula-

Table 4: *Precision Engagement Software Technologies*

Precision Engagement Activity	Advanced Software Technologies Needed
Information Dominance and Enhanced Situational Awareness	• Information Organization/Retrieval Using Ontology • Context-Sensitive Reference Architectures • Intelligent Software Agents • Model-Driven Computing • Human Factors Interactions With Complex Systems • Data Mining • Multi-Level Security
Precision Geo-Location of Time-Sensitive Targets	• Intelligent Software Agents
Shorter Sensor-to-Shooter Engagement Chain	• Context-Sensitive Reference Architectures • Model-Driven Computing
Wide Range of Precision Effects in Any Weather	• Software Technologies Are Adequate

Table 5: *Mature and New Software Technologies*

Mature, Deployed, Software Technologies	Advanced Software Technologies Supporting Protection of America
• High-Level Programming Languages • Compilers • Operating Systems • Object-Oriented Technologies • Relational Databases • Internet • Transmission Control Protocol/ Internet Protocol-Based Layered Networking • XML (Extensible Mark-Up Language)	**Technologies for Development of System Architectures** • Information Organization/Retrieval Using Ontology • Context-Sensitive Reference Architectures • Model-Driven Architecture Development • Reference Architecture Frameworks and Associated Development Processes **Other Advanced Software Technology Needs** • Human Factors Interactions With Complex Systems • Data Mining • Intelligent Software Agents • Cognitive Computing Techniques • Better Collaboration Tools

tion analysis provide metrics that can be used to eliminate, aggregate, or validate the key components and relationships with the family of architectures, using information organized via taxonomies and associated ontology. The reference architecture is continually updated and refined based on this feedback loop. The reference architecture is not the final blueprint for implementing systems-specific design and integration, but rather a reference of concepts providing the enabling cornerstone upon which systems can be empowered with large-scale mission capabilities. It is up to the organization accomplishing a software systems task to engineer and build an instance of the reference architecture to suit the needs of a particular domain, while maintaining compatibility with the overall standard reference architecture.

Reference architecture can be considered to have four abstract aspects: social, cognitive, information, and physical. Each aspect provides the context upon which to view system instances. Collections of systems instances change over time. The dynamics of a real-world environment necessitate the flexibility inherent in reference architecture to take into account changing elements over time.

The combination of the reference architecture and the four domain aspects provides the basis for examining mission systems in three dimensions instead of the traditional two as presented by the Department of Defense Architecture Framework (DoDAF)[2]. This three-dimensional view provides the basis for systems interoperability in a logical and meaningful way. Further analysis makes apparent the relationships between data, information, knowledge, and understanding required for combined systems operations and efficient management of available communications resources.

The mission-system reference architecture has the following properties:

- Provides the conceptual framework for specifying the four aspects (social, cognitive, information, and physical) of systems within the bounds of operational, system, and technical views prescribed by DoDAF.
- Acts as a template to guide domain-specific implementations of distributed network-centric systems while allowing a variety of design solutions.
- Defines the ontology for discussion and analysis purposes.
- Defines a complete set of architectural elements with well-defined interactions, functionality, and relationships with themselves and the external context.
- Defines how the elements communicate with each other, the basic operations associated with each element, and the nature of the communication.

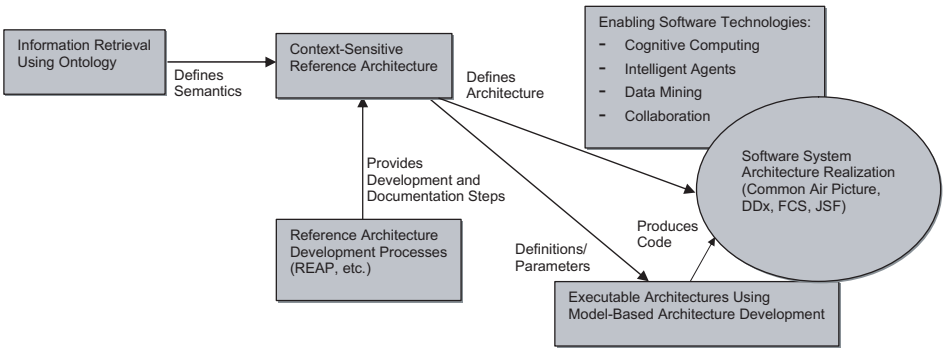


Figure 1: *Key Software Architecture Technologies Interact to Support Large Mission Systems*

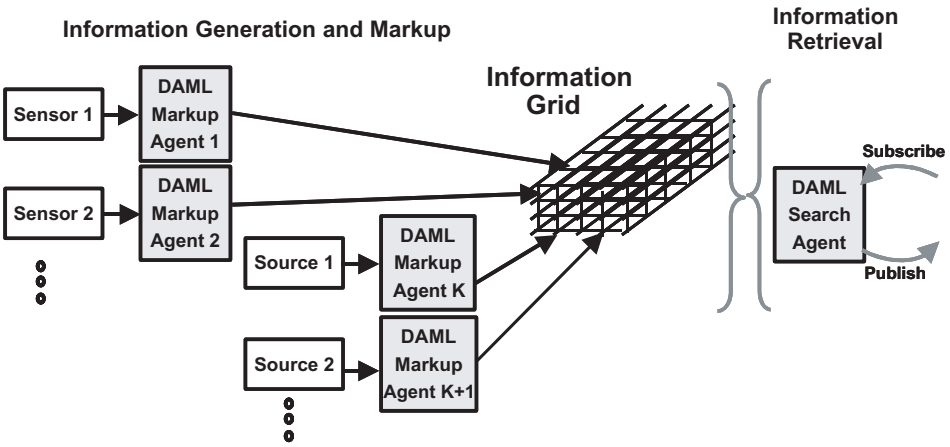


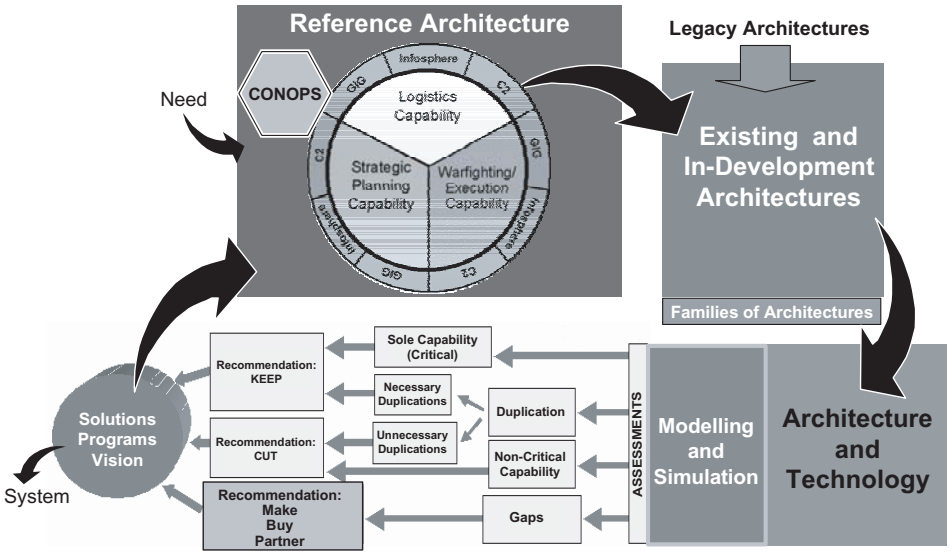
Figure 2: *Ontology-Based Information Retrieval*

Enterprise Reference Architecture Processes

The U.S. government has established direction and expectation for how complex systems of the future will be developed and integrated – via an ever-increasing emphasis on the importance of formalized architecture and enterprise architecture. Many aerospace and information technology companies are now developing and maturing their architecting processes to meet their business needs.

Lockheed Martin deploys its Architecture-Based Design and ARQuest Blueprint. Northrop Grumman has its Information Systems Architecture Analysis Continuum. IBM has the Enterprise Architecture Method. Boeing and General Dynamics promote their open systems architecture frameworks,

Figure 3: *Reference Architecture Application to Domain-Specific Instances*



Bold Stroke and OpenWings, respectively. Government, industry, and academia are establishing consortia, certification programs, and graduate curriculum to address the educational needs of this new discipline.

The system architecting process that Raytheon uses is known as Raytheon Enterprise Architecture Process (REAP) [1]. It extends a traditional focus on technical architecture to include business architecture, providing a comprehensive view across the enterprise. The REAP defines an end-to-end architecture process based on industry and government standards, including The Open Group Architecture Framework[3] Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance/Department of Defense Architecture Framework[4], Zachman Framework for Enterprise Architecture[5], and the Software Engineering Institute's Architecture Trade-off Analysis Method℠ (ATAM℠)[6].

Components

There are established industry and government standards to help address enterprise-wide architectural alignment among customer mission, business rules, data, application systems, organization, and technology. The primary standards unified within Raytheon's architecture process and other architecture processes to fulfill the components noted above are the following:

- **Methodology:** The Open Group Architecture Framework (TOGAF), Enterprise Edition.
- **Products:** DoDAF, final draft Zachman Framework for Enterprise Architecture.
- **Formats:** Unified Modeling Language[7], Integrated Computer-Aided Manufacturing Definition[8], DoDAF templates.
- **Validation:** ATAM.

It is important to note that although there are several integrated frameworks, they each address very different elements of the overall architecting process and their interrelation is both necessary and complementary.

Activities

Architecture processes are comprised of five primary activities: enterprise understanding, architecture planning, business architecting, technical architecting, and architecture validation. These activities are iterative in nature, internally and externally to the other.

In Raytheon's case, the five activities act as a wrapper around the phases of TOGAF's Architecture Development Method (ADM), providing supplemental guidance and describing its relationships to other standards. These subprocesses extending the TOGAF ADM include those for customer-focused architecting, quality attribute analysis, architecture concordance/configuration/consolidation, DoDAF product generation, ATAM, and quality attribute assessments. The completion of these activities results in a validated architecture package describing the enterprise from a variety of viewpoints or perspectives.

"When proved successful, model-driven computing has the potential to revolutionize the current means of systems specification, development, testing, and maintenance."

Model-Based Computing

Model-based computing is the term for system and software development that is driven and centered on models. These models are used to specify systems and software architecture, and low-level system design details. The models provide the means to translate the specified systems architectural artifacts defined via system architecture development processes into constituent platform-specific and platform-independent components. The concept of developing platform-independent models, followed by platform-specific models is quite powerful and allows our programs to migrate models to new computing hardware with minimal impact. Platform-independent models can also be used in multiple environments such as simulations, using the same system model.

This concept has been standardized via the Object Management Group (OMG)[9] in the Model-Driven Architecture initiative. The OMG is working to standardize these concepts in order to promote tool development and interoperability. Recently, the OMG has also formed an interest group specifically focused on standards for model-driven development of embedded software. This interest group will leverage recent significant advances made possible in large part via the leadership, insight, and funding support from DARPA. These new tools and technologies are laying the necessary foundations upon which the systems of the future will be specified, developed, tested, and maintained.

DARPA has been advancing the state-of-the-art application of model-driven computing to distributed, real-time, and embedded (DRE) systems. DARPA, via the Model-Based Integration of Embedded Systems (MoBIES)[10] program, is establishing an open-source, standards-based tool suite needed to accomplish the program's objectives. One MoBIES technology developer is the Institute for Software Integrated Systems (ISIS) at Vanderbilt University[11]. ISIS, as well as being a major contributor to the MoBIES program, is working to see that DARPA-funded efforts migrate into the mainstream. They are working to migrate DARPA-funded tools to the Eclipse Open-Tool Integration Framework via sponsorship from IBM.

Raytheon and the aerospace industry are actively involved with the development of standards that impact the future of model-driven computing within the OMG. These standards may be impacted by the further evolution of DARPA-developed tools and technologies from MoBIES and other DARPA programs. The maturation of those tools is being supported via membership in the newly formed Embedded Systems Consortium for Hybrid and Embedded Research.

Model-driven computing has had some noteworthy successes despite being used in limited domains. Two popular examples are The Mathwork Company's Matlab/Simulink®[12] and National Instruments' LabVIEW[13]. These pioneering tool suites demonstrate that model-driven computing is effective in limited application domains. Until recently, modeling of the entire system, middleware, and application, needed to be accomplished for each system. This made it cumbersome, time-consuming, and expensive to develop effective models. It was not until the separation of the application from the middleware, and models of the middleware could be shared and leveraged, that model-driven computing has come into its own.

Additional advances in model-driven computing are necessary before it can become commonplace in DRE systems development. Scalability in both breadth and depth of model-based computing must be addressed. When proved successful, model-driven computing has the potential to revolutionize the current means of systems specification, development, testing, and maintenance. We expect that the most significant impact will be

realized in system verification. With complete and executable system models that are independent of the hardware platform, system verification will move forward in the development process, reducing the cost and risk of errors, and facilitating the final system verification effort.

Conclusion

William Gibson once stated, "The future is already here; it is just unevenly distributed" [2]. The successful implementation of the large systems of systems needed for America's protection will be expedited by using emerging, but not yet widely deployed, software approaches that support the development of robust system architectures. The key technologies of ontology, context-sensitive reference architectures, architecture definition processes, and model-based computing are beginning to be integrated to develop robust systems that are key for America's defense. More research is required to make these approaches scalable and capable of integrating with existing systems, but the foundations exist today.◆

Acknowledgements

We would like to acknowledge the contributions of Steve Ignace, Rolf Siegers, Mike DaBose, Chris Grounds, Ralph Woods, Tom Flynn, Bryan Lail, Edwin Lee, Bhatra Patel, Doris Tamanaha, Ron Williamson, and Don Wilson.

References

1. Siegers, R. "The Raytheon Enterprise Architecture Process." INCOSE 2003, Crystal City, VA., July 2003.
2. DeLong, J. Bradford. "The Real Shopping-Cart Revolution." Wired Mar. 2003 <www.wired.com/archive/11.03/view.html?pg=5>.

Notes

1. See <www.daml.org>.
2. See <http://deskbook.dau.mil/software/gen/comparch-def.html>.
3. See <www.opengroup.org/architecture/togaf>.
4. See <www.dod.mil/comptroller/bmmp/pages/arch_arch_home.html>.
5. See <www.zifa.com>.
6. See <www.sei.cmu.edu/ata/ata_ method.html>.
7. See <www-306.ibm.com/software/rational/uml>.
8. See <www.idef.com/default.html>.
9. See <www.omg.org>.
10. See <www.rl.af.mil/tech/programs/MoBIES>.
11. See <www.isis.vanderbilt.edu>.
12. See <www.mathworks.com>.
13. See <www.ni.com/labview>.

About the Authors

Gregory S. Shelton is vice president of Engineering, Technology, Manufacturing, and Quality for Raytheon Company. He is responsible for developing and implementing enterprise engineering, quality and program management processes and tools, and integrating technology strategies, road maps, and competitive assessments. In 2002, he was elected associate fellow for the American Institute of Aeronautics and Astronautics. Shelton has a bachelor's degree in electrical engineering from California Polytechnic University and a master's degree in engineering and management from the University of California, Los Angeles.

Raytheon Global Headquarters
870 Winter ST
Waltham, MA 02451
Phone: (781) 522-3000
Fax: (781) 522-3001
E-mail: gshelton@raytheon.com

Randy Case is technical area director at Raytheon Company for Architectures and Systems Integration, Garland, Texas. He was the architect of the Raytheon Integrated Product Development System. Case has worked on projects that span the entire life cycle from independent research and development to operational support. He is co-chair of the International Council on Systems Engineering Standards Technical Committee, and has contributed to a number of systems-related standards. Case has a Bachelor of Science in electrical engineering from the University of Texas at Arlington.

Raytheon/Intelligence and Information Systems
1200 South Jupiter RD
Garland, TX 75042
Phone: (972) 205-5306
Fax: (972) 205-8083
E-mail: randy_r_case@raytheon.com

Louis P. DiPalma is the manager of the Integrated Warfare and Sensor Systems Software Department of the Raytheon Integrated Defense System Integrated Software Development and Human Systems Interface Engineering Center. DiPalma has been involved in the design and development of submarine combat control systems and weapon launching programs, as well as several fire control systems for surface combatants. His focus has been centered on the infusion of new technology into the Raytheon integrated decision-support product line, with a primary focus on naval combat systems.

Raytheon Integrated Defense Systems
1847 West Main RD
Portsmouth, RI 02871
Phone: (401) 842-5592
Fax: (401) 842-5232
E-mail: louis_p_dipalma@raytheon.com

Dan Nash is director of Software Engineering, Raytheon Corporate Engineering, Waltham, Mass. He is responsible for coordinating mission-critical software engineering activities across the various business units, as well as other assignments such as Red Teams. His recent work has been in the areas of the Capability Maturity Model®, Integration and software supplier management. He holds a Bachelor of Science in electrical engineering from North Carolina State University.

Raytheon Company
870 Winter ST
Waltham, MA 02451-1499
Phone: (781) 522-3362
Fax: (781) 522-6434
E-mail: j_dan_nash@raytheon.com

Bridging Agile and Traditional Development Methods: A Project Management Perspective

Paul E. McMahon
PEM Systems

Monday, 19 April 2004
Track 4: 3:35 - 4:20
Ballroom D

Today, companies are reporting success in meeting rapidly changing customer needs through agile development methods. Many of these same companies are finding they must collaborate with organizations employing more traditional development processes, especially on large Department of Defense projects. While it has been argued that agile methods are compatible with traditional disciplined processes, actual project experience indicates conflicts can arise. This article identifies specific project management conflicts that companies face based on actual project experience, along with strategies employed to resolve these conflicts and reduce related risks. Rationale, insights, and related published references are provided along with lessons learned and recommendations. If you work for a company that is using or considering using agile development, or your company is collaborating with a company using an agile method, this article will help you understand the risks, issues, and strategies that can help your project and organization succeed.

This article was motivated by a case study where a small company using a well-known agile method – eXtreme Programming (XP) – requested help addressing specific conflicts that arose on the project where they were a subcontractor to a larger organization employing a traditional development method. The purpose of this article is not to compare agile and traditional methods, but to raise awareness of potential project management conflicts that can arise when a company employing an agile method collaborates with a company employing a traditional development methodology. It also identifies practical steps that can be taken to reduce related risks.

It is worth noting that the case study presented is not unique. Published references documenting similar conflicts are provided. Also notable is that the motivation for examining this project extends beyond the case study itself. Today there exist increasing opportunities for small companies to gain new work through software outsourcing from traditional development organizations.

Where Are We Going?

In this article, I first identify key case study facts along with relevant information and common misperceptions related to traditional and agile methods. Next, I identify four conflicts observed along with five recommendations and one lesson learned. The company named SubComp refers to the subcontractor employing an agile methodology. The company named PrimeComp refers to the prime contractor employing a traditional development methodology.

Case Study Key Facts

Shortly before I was asked to help SubComp, PrimeComp's customer had withheld a progress payment based on a perceived risk observed at a recent critical design review (CDR). Written comments provided to PrimeComp indicated that the customer wanted to see working software in order to *assess the proposed design and related risk*. The area of concern was SubComp's responsibility. Upon receiving the customer comments, PrimeComp requested that SubComp provide additional detailed design documentation.

It is important to note that PrimeComp required all correspondence between the customer and SubComp to go through them. It is also important to note that most of the contractually required documentation was not formally due until the end of the project, and, prior to the CDR, little had been communicated to SubComp with respect to documentation content and expectations. The project was planned using a traditional waterfall life cycle with a single CDR.

Early in the project, SubComp had identified multiple technical risks. However, it had decided in the early stages to focus its small agile team on a single technical risk that it had assessed to be of much greater significance than all other risks. At the recent CDR, SubComp had provided a demonstration with working software that addressed this risk to the customer's satisfaction.

The risk the customer was currently raising was one SubComp viewed as lower priority. To address this risk, the XP team was focusing on a second demonstration with working software to show to the customer at a follow-up CDR. In parallel, it was also driving to meet PrimeComp's request for additional detailed design documentation. This follow-up CDR had not originally been planned, and it was causing project tension because of the progress payment holdup.

Agile Development Methods

In the spring of 2001, 17 advocates of agile development methods gathered in Utah and agreed to a set of four values and 12 principles referred to as the Agile Manifesto [1]. The four values are expressed in the following:

> We are uncovering better ways of developing software by doing it and helping others do it … Through this work we have come to value:
> - Individuals and interactions over processes and tools.
> - Working software over comprehensive documentation.
> - Customer collaboration over contract negotiation.
> - Responding to change over following a plan.
>
> That is, while there is value in the items on the right, we value the items on the left more. [1]

One misperception of agile methods is that they hold little or no value in documentation and plans. Note that the value statements express a relative value of documentation and plans with respect to working software and responding to change.

Traditional Development Methods

The traditional waterfall model is well known, but it is important to understand that it is not an incremental model. Today, Department of Defense policy 5000.1 and 5000.2 [2] strongly encourages using the spiral model for software development. Although the spiral model was first introduced by Barry Boehm in the mid-80s [3], the risk-driven essentials of the model frequently have been misunderstood. To help clarify, Boehm recently identified six spiral essentials [2]:

- Concurrent determination of key artifacts.
- Stakeholder review and commitment.
- Level of effort driven by risk.
- Degree of detail driven by risk.
- Use of anchor-point milestones.
- Emphasis on system and life-cycle artifacts (cost/performance goals, adaptability).

A Perspective on Waterfall, Spiral, and Agile Development

Historically, the waterfall life-cycle model has been closely associated with heavyweight documentation. The spiral model has also historically been misinterpreted as an incremental waterfall model, rather than as a risk-based model as clarified by Boehm. It is important to note the focus on people (individuals, stakeholders), products (working software, key artifacts), and change (responding to change, adaptability) common to both the Agile Manifesto and the spiral essentials.

Methods Compatibility or Conflict?

It would seem from this observation that a company using an agile methodology should be able to successfully collaborate with a company using a traditional development method, especially if the project contained risk. To further support this position, Mark Paulk, co-author of the Software Engineering Institute's Capability Maturity Model® (CMM®), which has been associated with rigorous traditional development methods, has stated, "XP is a disciplined process, and the XP process is clearly well defined. We can thus consider CMM and XP complementary" [4].

Despite this evidence of methods compatibility, a different situation appears to exist in the developmental trenches. This is clearly pointed out by Don Reifer in the following statement made in reference to one of his own studies:

> Instead of trying to make XP work rationally with the firm's existing processes, each side is pointing fingers at the other. No one seems to be trying to apply XP within the SW-CMM context rationally and profitably as the sages have suggested ... XP adherents feel they don't have time for the formality … Process proponents argue … quality will suffer and customer satisfaction will be sacrificed. [5]

One proposal to address this conflict has been put forth by Scott Ambler in the form of a *blocker* who runs *interference* for the team by providing, in Ambler's words, "the documents that the bureaucrats request" [6]. The term blocker essentially means someone whose sole job is to keep non-agile project stakeholders from hindering the agile development team's progress.

"Using an incremental life-cycle model is critical because many of the conflicts observed are rooted in the all up-front thinking that comes with the single-increment waterfall model. Incremental thinking is fundamental to agile methods and crucial to bridging the two methods."

In the following paragraphs we identify four conflicts observed on the PrimeComp case study project.

Conflict 1: Working Software vs. Early Design Documentation

Part of the difficulty faced by SubComp is the conflict between what they perceive the end-customer wants with respect to risk management, and what is being asked for by their immediate-customer, PrimeComp. The end-customer has asked to *see working software*. It appears that the end-customer wants more than a *paper* design to assess the risk, yet PrimeComp is asking SubComp to provide more detailed design documentation.

Conflict 2: Single vs. Multiple-Increment Life Cycle

Assuming SubComp succeeds in addressing the immediate high visibility risk, what if yet another risk pops up at the follow-up CDR? Will there be a follow-up to the follow-up CDR with further delays of progress payments?

Agile teams often tackle tough issues first, as did SubComp. They focus on achieving customer satisfaction through frequent software deliveries based on clear priorities. Agile teams are also usually small and often do not have adequate resources to work multiple risks in parallel. The single iteration through the waterfall model with the planned single CDR milestone was a major project management conflict for the agile team. From a project management perspective, it was a critical conflict since a significant payment was withheld.

Conflict 3: Formal Deliverable Documentation Weight

Hearing that the documentation was not due until the end of the project led me to ask two questions:

- What exactly were the contractual documentation requirements?
- Did SubComp know PrimeComp's documentation expectations?

Companies that employ agile methods tend to provide *lightweight* documentation. This is, at least partially, because they value working software more than documentation. Although large documents are not a requirement of traditional development methods, cultural expectations often tend to the *heavyweight* side.

Waiting to deliver documentation until late in the project creates a potential conflict, especially if expectations have not been set through early communication. Because of the stress being placed on the agile team to complete the demo software and to provide additional detailed documentation, the possibility of using a blocker was considered.

Conflict 4: On-Site Customer Collaboration

Agile methods *embrace* [7] changing requirements, even late in development. During my discussions with SubComp personnel, I discovered that the contractual project requirements were, in the words of one team member, "high level and

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

open to many interpretations." When contractor and customer work closely on an agile development project, embracing change is eased through close collaboration. When a prime contractor inserts itself in the middle, effective collaboration can be stifled, creating additional conflict and risk.

During my discussions with SubComp personnel, one member of the agile team said,

> What is difficult from my perspective is that the details they [PrimeComp] are asking for are the lowest risk and lowest value technically … spending time on what they feel they need puts our small team at risk for actually completing the product, which is what ultimately matters, I think.

This statement led me to stop and consider just what ultimately did matter. The agile team was addressing the technical risks in a planned manner, but a key ingredient to effective XP operation was missing. How could the agile team determine and act based on what ultimately mattered to the customer when the customer was not collaborating closely on-site?

Given these four observed conflicts, what strategies make sense and what can be done to bridge agile and traditional development?

Recommendation 1: Plan Collaboratively and Use an Incremental Life Cycle

Some believe those who use agile methods do not follow a plan. This is a misunderstanding. Planning is actually a core principle of agile methods [7]; however, agile teams tend to plan in smaller time increments and more frequently than those who use traditional methods. This distinction has been clarified by the characterization of XP as *planning driven* rather than *plan driven* [8]. Planning takes place with both methods, but with agile methods the focus is on the act of planning, rather than a document.

I recommend for similar projects that the initial planning be done collaboratively, with the prime contractor, subcontractor, and customer working closely. I also recommend that an incremental life-cycle model be employed to aid in aligning the agile subcontractor's work within the overall project schedule. Using an incremental life-cycle model is critical because many of the conflicts observed are rooted in the all up-front thinking that comes with the single-increment waterfall model. Incremental thinking is fundamental to agile methods and crucial to bridging the two methods.

In the case study, it was too late to replan the project with an incremental life cycle, but I did recommend that SubComp step back and re-evaluate their strategy to the upcoming follow-up CDR. The following questions needed to be answered:

- Were there other risks that the customer felt had to be addressed at this point for the project to move forward successfully?
- What else would it take to close the CDR?

We had to find out the customer's CDR completion criteria, but we had to do it within an agile mind-set. That is, quickly and with minimal resources given that the small agile team was already overextended working to complete the demo and the prime contractor's request for additional documentation.

"The frequent feedback from multiple spirals can help your risk mitigation visibility and ultimately your project's success."

Recommendation 2: Use the Spiral Model and Well-Defined Anchor-Point Milestones to Address Risk

If one of the risk-responsible collaborating team members is employing an agile method, a spiral model with well-defined anchor points [2, 9] can go a long way to reduce potential project conflict and risk. This model can help the traditional development prime contractor as well as the agile subcontractor.

Providing working software early to address high-risk areas makes sense, but it is not sufficient to meet all project management needs. If you are the prime contractor, you want to make sure all the risks are being addressed in a timely and prioritized fashion. The frequent feedback from multiple spirals can help your risk mitigation visibility and ultimately your project's success.

From the agile subcontractor's perspective, you want your team to be able to focus and solve the highest priority risks early, but you also want to know what the prime contractor's expectations are along the way. By agreeing to the anchor-point milestones during the early collaborative planning activity, expectations can be made clear on both sides, allowing the project to operate more effectively. One of the reasons an incremental life-cycle model is recommended is because it often leads to earlier communication concerning priorities and risks. When a traditional waterfall life-cycle model is selected, early discussions concerning priorities are often missed.

Recommendation 3: Plan for Multiple Documentation Drops and Use a Bridge Person

One reason collaborative initial planning is recommended is to get discussions going early concerning product deliverables thereby reducing the likelihood of late surprises. In the case study, it was decided not to use a *blocker* to provide the contractual documentation. Discussions led the agile team to the conclusion that the blocker notion brought with it a negative view of documentation. The blocker was seen as someone outside the agile team whose job it was to develop the documentation without *bothering* the team. This was not consistent with SubComp's view of documentation, nor was it consistent with the values expressed in the Agile Manifesto.

Our solution was to use what we called a *bridge* person, rather than a blocker. The bridge person, unlike the blocker, was a member of the agile team who participated in team meetings providing a valuable service to the team by capturing key verbal points and whiteboard sketches thereby providing useful maintainable *lightweight* documentation that would ultimately help both contractors and the customer.

I recommended that multiple drops of documentation be provided to PrimeComp prior to the contractual delivery date to get early feedback and reduce late surprises. Waiting until late in the game to deliver documentation is risky, especially when expectations are uncertain.

Recommendation 4: Find a Way to Make Customer Collaboration Work for Effective Requirements Management

The reason establishing a close collaborative working relationship with the customer was not easy in our case study was because PrimeComp was sensitive to any contact between the end-customer and SubComp. This sensitivity was, at least partially, motivated by the desire to maintain control. It is also possible that uncertainty surrounding how the end-customer would perceive the use of an agile methodology fueled PrimeComp's desire to maintain a separation between

SubComp and the end-customer.

A recommendation I would make today to a prime contractor facing similar situations is to recognize that the key to maintaining real project control is the management of risks associated with the subcontractor's effort. One of the most common risks in similar situations is requirements creep, which often fails to get recognized until late in the project's test phase when the customer starts writing new discrepancies because the product does not meet what they now perceive the requirements to mean.

This situation frequently occurs because the end-customer and product developer (agile subcontractor) fail to collaborate sufficiently in the early stages reaching common agreements on potentially ambiguous requirements statements. In an agile development environment, this risk increases because requirements are often written as *story cards* [7], which have an implicit dependence on face-to-face communication to resolve potential differing requirements interpretations.

One common misperception of agile methods is that the requirements are not controlled since they are allowed to change late in the game. This is a legitimate concern that could also fuel why a prime contractor might want to keep an end-customer away from an agile subcontractor, but it also demonstrates a fundamental misunderstanding of agile methods.

As Craig Larman explains, "Iterative and agile methods embrace change, but not chaos" [10]. The following rule clarifies the distinction: "Once the requests for an iteration have been chosen and it is under way, no external stakeholders may change the work" [10].

If you are a prime contractor, I recommend that you check with your agile subcontractor to ensure they understand this crucial distinction between embracing change and living in chaos. I also recommend that a member of the prime contractor's team be placed on the agile team. Then encourage and support as much customer collaboration as you can between the agile team and the end-customer to help manage your own risk.

If you are an agile subcontractor, you want to demonstrate to the prime contractor that you are effectively managing your allocated requirements. Those employing agile methods often use *story point* [11] charts to depict work remaining and requirements creep. While story points and anchor points are different, they can be used together to help bridge the two methods.

Anchor points can be viewed as spiral model progress checkpoints [2, 9]. One weakness with traditional approaches has been the accuracy of the methods employed to measure progress. Story points provide an objective progress-measurement method based on stories verified through successfully completed tests [11].

At the start of each increment, I recommend that the agile subcontractor provide the prime contractor with a documented list of agreed-to stories to be completed in the upcoming increment. Story point charts can then be used to objectively back up anchor-point progress assessments, leading to improved team communication and trust.

Recommendation 5: Document and Communicate Your Process

I recommended to SubComp that they put together a presentation documenting their agile process from the project management perspective. This presentation would include key terms, roles, and responsibilities. Terms unique to the agile method such as coach, tracker, and metaphor [7] would be mapped to traditional terms such as project manager and architecture.

> *"If you are an agile subcontractor, you want to demonstrate to the prime contractor that you are effectively managing your allocated requirements."*

Recommendations for incremental lifecycle model, contract deliverables, and reviews compatible with both agile and traditional development methods should be included. Key to the presentation is a description of how SubComp's agile method fits within a traditional project management framework using a spiral model focusing on risk management. I then recommended to SubComp that they take every opportunity to communicate the key points in the presentation to PrimeComp, the end-customer, and to other traditional development contractors who might hold potential new business opportunities through software outsourcing.

Lesson Learned

When on-site customer collaboration exists, conflicts associated with vague requirements can often be resolved quickly. However, when the customer is not easily accessible, an agile subcontractor with vague requirements can quickly be placed at great risk.

We have learned that today's multi-contractor collaborative projects often do not lend themselves well to full-time, on-site customer collaboration. However, this does not mean that these projects cannot benefit from agility.

In such cases, I recommend a hybrid agile method with a focus on a more traditional requirements development and management method. Successful hybrid agile methods are not new [8, 12]. Keep in mind that hybrid does not imply all requirements up-front, but it does imply that once an iteration is under way, requirements must remain fixed to avoid chaos.

Conclusion

Today, we know how to manage geographically distributed teams formed from companies with divergent cultures and experiences [13]. Bridging agile and traditional development is the next step for organizations looking to take advantage of increasing new business opportunities through collaboration.

If you are experiencing conflicts similar to what has been described in this article, first examine your lines of communication. Look at your terminology. Are you communicating effectively what it is you do and how you do it? If you heard recently that a customer review did not go well, consider that the cause could be as simple as your agile terminology not connecting to the ear of a listener familiar only with traditional methods.

Allowing late requirements changes can work when your customer is on-site working next to you. But if you do not have an on-site collaborative customer, consider a hybrid approach to avoid major trouble late in the project.

Consider bridging, rather than blocking to meet milestone deliverables. More importantly, consider communicating to your collaborative partners and customers through examples of your products to gain their buy-in early, including the weight of your proposed documentation. Let them know that being agile is not cheating, but is in the best interests of everyone.

While many of the solutions described in this article are similar to those employed on non-agile projects, these solutions should not be taken for granted for two reasons. First, too often on hybrid agile-traditional projects, we find emotion

Coming Events

June 2-4
Symposium on Access Control Models and Technologies 2004
Yorktown Heights, NY
www.sacmat.org

June 11-13
ACM Sigplan 2004 Conference on Language Compilers and Tools for Embedded Systems
Washington, DC
http://lctes04.flux.utah.edu

June 14-17
SEPG Europe 2004
London, England
www.espi.org/frm–sepg.asp

June 14-18
ICSPI 2004 International Conference on Software Process Improvement
Washington, DC
www.icspi.com

June 23-26
Agile Development Conference 2004
Salt Lake City, UT
www.agiledevelopment conference.com

June 27- July 2
USENIX Annual Technical Conference
Boston, MA
www.usenix.org/events/usenix04

July 21-25
CITSA 2004 International Conference on Cybernetics and Information Technologies, Systems, and Applications
Orlando, FL
www.infocybernetics.org

April 18-21, 2005
2005 Systems and Software Technology Conference
Salt Lake City, UT
www.stc-online.org

getting in the way of clear thinking, often leading to a fundamental breakdown of communication. Second, we are learning through agile methods more effective techniques to measure progress and communicate. As Robert Martin pointed out in referring to agile methods:

> They're not a regression to the cave, nor are they anything terribly new: Plain and simple, the agile bottom line is the production of regular, reliable data – and that's a good thing. [11]

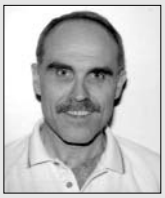
Don Reifer said, "No one seems to be trying to apply XP within the SW-CMM context rationally and profitably as the sages have suggested" [5]. In our case, studying the proactive steps taken based on the recommendations led to a successful follow-up CDR and to improved communication and early documentation agreements. Today, SubComp recognizes the value of XP, but they also recognize the value and need for fundamental project management, and they are looking to the CMM Integration℠ framework [14] to help guide related improvements.

Agility is not counter to effective project management, but agile methods do not provide all of the project management needs necessary for success. Wrap your agile development process in a lightweight project management framework, and watch your communication and collaboration improve and your project and company succeed.◆

References

1. Cockburn, Alistair. Agile Software Development. Addison-Wesley, 2002: 215-218.
2. Boehm, Barry. "Spiral Model as a Tool for Evolutionary Acquisition." CrossTalk May, 2001 <www.stsc.hill.af.mil/crosstalk/2001/05/index.html>.
3. Boehm, Barry. A Spiral Model of Software Development and Enhancement. Proc. of An International Workshop on Software Process and Software Environments, Trabuco Canyon, CA., Mar. 1985.
4. Paulk, Mark. "Extreme Programming from a CMM Perspective." IEEE Software Nov./Dec. 2001: 19-26.
5. Reifer, Don. "XP and the CMM." IEEE Software May/June 2003: 14-15.
6. Ambler, Scott. "Running Interference." Software Development July, 2003: 50-51.
7. Beck, Kent. Extreme Programming Explained: Embrace Change. Addison-Wesley, 2000.
8. Boehm, Barry, and Richard Turner. Balancing Agility and Discipline: A Guide for the Perplexed Addison-Wesley, 2003: 33-34, 233.
9. Boehm, Barry, and Daniel Port. "Balancing Discipline and Flexibility With the Spiral Model and MBASE." CrossTalk Dec. 2001 <www.stsc.hill.af.mil/crosstalk/2001/12/boehm.html>.
10. Larman, Craig. Agile and Iterative Development: A Manager's Guide. Addison-Wesley 2003: 14.
11. Martin, Robert C. "The Bottom Line." Software Development Dec. 2003: 42-44.
12. McMahon, Paul E. "Integrating Systems and Software Engineering: What Can Large Organizations Learn From Small Start-Ups?" CrossTalk Oct. 2002: 22-25 <www.stsc.hill.af.mil/crosstalk/2002/10/mcmahon. html>.
13. McMahon, Paul. E. Virtual Project Management: Software Solutions for Today and the Future. St. Lucie Press, 2001.
14. CMMI Product Team. Capability Maturity Model® Integration (CMMI®), Version 1.1. Pittsburgh, PA: Software Engineering Institute <www.sei.cmu.edu>.

About the Author



Paul E. McMahon, principal of PEM Systems, provides technical and management services to large and small engineering organizations. He has taught software engineering at Binghamton University; conducted workshops on engineering process and management; and published over 20 articles, including articles on agile development and distributed development in CrossTalk, and a book on collaborative development, "Virtual Project Management: Software Solutions for Today and the Future." McMahon also presented at the 2003 Software Technology Conference on "Growing Effective Technical Managers."

PEM Systems
118 Matthews ST
Binghamton, NY 13905
Phone: (607) 798-7740
E-mail: pemcmahon@acm.org

Understanding Software Requirements Using Pathfinder Networks

Udai K. Kudikyala and Dr. Rayford B. Vaughn Jr.
Mississippi State University

Thursday, 22 April 2004
Track 6: 10:15 - 11:00
Room 150 D - F

Understanding and communicating user requirements early in the software development life cycle is essential for satisfying user needs as well as reducing defects, cost, and schedule. This article reports on a technique that uses pathfinder networks to discover and evaluate mental models that represent stakeholders' perception of software requirements. The results obtained by applying this technique on multiple projects are also described.

During the initial phase of any system development activity, software developers are challenged to uncover, understand, and specify user requirements. It is important to have a common understanding among users/customers, project managers, and developers (collectively known as stakeholders) regarding requirements of the software system being developed. This is often considered a major risk factor in software development projects [1]. The sooner misunderstandings are resolved, the more likely developers will build a product correctly [2, 3, 4].

You might consider early requirements in the context of a mental model – or a representation of how a customer or developer thinks about a set of requirements and visualizes them as a whole. Mental models not only model an understanding of the system but also misconceptions that the stakeholder may have [5]. They are also used to understand and communicate what a user actually thinks [6]. Empirical evidence we have obtained indicates that such models may also be useful in identifying misunderstood, duplicate, and ambiguous requirements.

For the past three years, we have experimented with a technique known as pathfinder networks (PFNETs), which are more fully described in the next section. This technique comes from the field of artificial intelligence and one of its properties is the ability to represent knowledge structures as they exist in the minds of humans. This representation is formulated through a facilitator working with one or more subjects who take concepts and group them in terms of their relatedness.

In our research, we used software requirements as *concepts* and applied the PFNET technique to a requirements document separately for both the developer and customer. In both cases, the resulting PFNET was considered a *mental model*, and the two mental models were compared mathematically to determine how close they were. We also discovered this technique was useful in identifying redundant, ambiguous, and misunderstood requirements.

While a complete discussion of the mathematics of the process is beyond the scope of this article, we do provide references and contact information for the interested reader. We have implemented the required mathematics in working software packages and have used the technique on two industrial experiments for real-world projects. In all cases, the PFNET technique was successful in identifying requirements misunderstandings and contributed to a better understanding between the developer and customer early in the life cycle.

> *"Mental models not only model an understanding of the system but also misconceptions that the stakeholder may have. They are also used to understand and communicate what a user actually thinks."*

The techniques we have developed can generally be learned and used in less than eight hours of training. We have not estimated the cost of this training, but believe it to be minimal. We have validated the utility of this technique for small to medium-size software development activities and now intend to publish the results and assist in transferring this technique to industrial use. We are especially grateful to AmerInd[1] Inc. and Nortel for their assistance in using and validating our PFNET work within their software development organizations.

PFNETs

The PFNETs [7] have been used widely to represent knowledge structures. They have been used to model the knowledge of experts and novices in the computer-programming domain [8] and to assess students' knowledge when compared to that of experts [9]. We have successfully applied this technique to the software-engineering domain and, in particular, to the problem of requirements verification and validation. The technique is briefly described in this article, but more detail can be found in [10, 11].

Essentially, we generate a PFNET for the customer/user community and a separate network for the developer community. These networks represent the current *mental model* of the requirements for both communities. By calculating the correlation between the two network models, we can then estimate the degree of common understanding. With additional analysis, we are also able to identify redundant and ambiguous requirements.

The PFNETs' original objective was to generate a network model from *psychological proximity data*, which is the subjective estimate of the closeness or relatedness between requirements as perceived by a stakeholder. The primary goal is to arrive at network representations with nodes representing requirements and links representing relations between requirements. Weights on the links represent the dissimilarity between the requirements. These edge-weights are calculated based on the categorization of information about the requirements that are collected from each stakeholder.

The process of categorization basically means that stakeholders are asked to place individual requirements into categories based on their perceived relatedness. The pathfinder algorithm is applied to concepts (requirements) from the computer science discrete structures domain. Figure 1 (see next page) reveals how the categorization of information about the

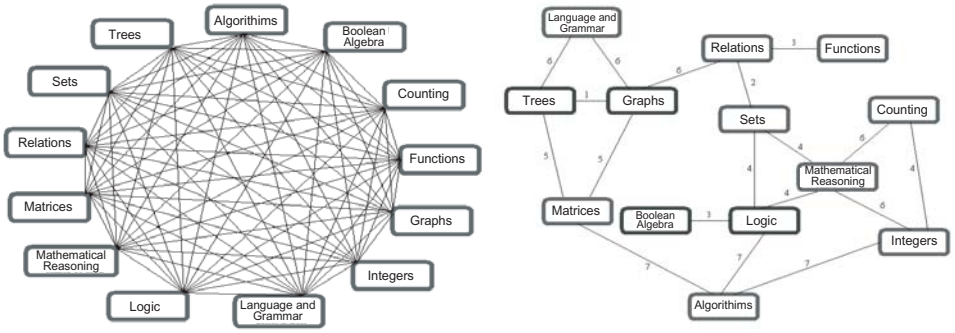


Figure 1: *A Fully Connected Graph With 13 Concepts and the Resulting PFNET With Only 18 Links*

concepts collected from a group of experts results in the generation of a conceptual structure represented by a PFNET.

Essentially, a link exists between a pair of nodes if and only if there is no shorter alternate path between that pair. The initial conceptual network is assumed to be a fully connected graph since no information is available about how the stakeholders relate the list of concepts. Therefore in Figure 1, the original conceptual structure (on the left) is reduced to a more understandable structure (on the right) by removing all links between concepts except for the shortest path or least dissimilar relationship.

Our assumption is that a stakeholder's requirements knowledge consists of the requirements and the interrelationships among those requirements. Figure 2 shows a portion of a PFNET that was generated in one of our experimental projects by the developer. The edge-weights represent the dissimilarity between the pairs of requirements. The lower the edge-weight the lower the dissimilarity and higher the similarity between the two requirements as perceived by the stakeholder.

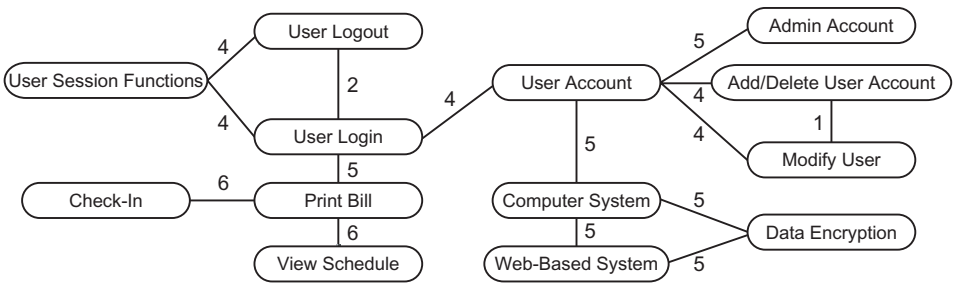
The similarity count for a pair of requirements is determined by the number of times that pair shows up together in the categorization of information collected from the stakeholders. A dissimilarity matrix is computed by subtracting each similarity count from the highest similarity count plus one. The pathfinder algorithm is then applied to the dissimilarity matrix to generate the PFNET. While details of these steps are not provided here, they are implemented in our software. Specifics can be provided to the reader on request.

In Figure 2, most of the developers perceived *Add/Delete User Account* and *Modify User* as closely related. We know this because the link weight is the minimum (least dissimilar), between the two requirements. A number of developers also perceived *User Account* and *Add/Delete User Account* to be related but to a lesser degree. The absence of a link between requirements could mean that very few or none of the developers perceived those requirements as being related. For example, *Admin Account* and *Add/Delete User Account* are not directly connected because the developer perceived *Admin Account* to be more closely related to *User Account* than to *Add/Delete User Account*.

In summary, the primary benefit of the PFNET is the ability to model aspects of human semantic memory. In our case, the ability to model the stakeholders' perception of requirements as graphs is useful. It provides the ability to mathematically evaluate and compare these networks for similarities and dissimilarities to reveal potential misunderstandings.

While this article represents only an overview of the PFNET technique, it is important to note that it is not new – it has been used in other applications outside software engineering for many years. Our work applies this technique to software engineering requirements analysis for the first time.

Figure 2: *Example of a PFNET (Partial) Generated for a Group of Developers*



We generated two PFNETs – one for the customer and one for developers. The categorization of information collected from each developer is combined, and a single consensus PFNET for that group is generated. A similar procedure is used for the end user. We then compare and analyze the two networks to determine the consistency of understanding between the two, as well as determine which individual requirements may not be well understood. Specific results of our application of this technique in projects are presented in the following sections.

Initial Experiment Results

Experiments were first conducted at Mississippi State University (MSU) using students taking a software engineering class [12]. Four real customers agreed to work with the class in order to obtain a needed software product. Students developed a software requirements specification (SRS) for each system, which was the basis for generating all subsequent PFNETs. The following procedures were used (implemented today in software):

- Requirements were individually extracted from each SRS. All participants were asked to read the SRS to become familiar with the requirements. Each student and customer then categorized requirements into groups based on their perceived relatedness. This was achieved by using index cards with requirements printed on each card. The set of index cards and a copy of the SRS were distributed to the customer and the developers working on that project. Instructions were given on the procedure for categorization; none of the participants were allowed to consult with each other during the categorization activity. The categorization activity consisted of grouping the index cards into piles of related categories. The relatedness decision was entirely up to the participant and was very subjective. We refer to the groups of index cards collected from each participant as the categorization of information.
- The categorization of information was then collected and a similarity matrix (N rows by N columns, where N is the number of requirements) was generated for each participant. When a pair of requirements appears together, the count in that cell of the similarity matrix is incremented by one. We refer to the similarity count as a co-occurrence count for that pair of requirements. To compute a consensus (group) PFNET, the corresponding elements of the similarity matrices for that group are simply added

to generate a consensus similarity matrix.
- Dissimilarity matrices were generated by subtracting each co-occurrence count in the similarity matrix from the maximum co-occurrence count plus one (to avoid a zero dissimilarity count in any cell).
- The dissimilarity matrices for the group of developers and the customer were then used as input to the Pathfinder generation program resulting in the generation of two PFNETs.
- The resulting PFNETs are then correlated with each other, producing a correlation coeffcient (cc) that is used to measure similarity between the mental models.

Adding substance to the preceding steps and realizing that several new terms were introduced to the reader, we provide a short example in the online version of this article at <www.stsc.hill.af.mil/crosstalk/2004/05/0405kudikyala.html>.

The cc ranges from -1 through +1, where -1 represents no similarity and +1 represents perfect similarity between the two networks [13]. For all projects, the following heuristics were applied:
- A cc of a network/node below 0.4 indicates little or no similarity.
- A cc from 0.4 through 0.7 indicates a moderate degree of similarity.
- A cc of more than 0.7 indicates very good to strong similarity.

The boundary values we assigned above are subjective and were selected based on empirical evidence. Table 1 shows the overall ccs between the developer and user PFNETs for the four projects in our experiment. In Table 2, each row shows the percentage of requirements with different ccs between developer and user PFNETs for each system developed.

Thus, the higher the value of the cc, the more similar the mental models of the customer and developers appeared to be at the early stages of product development. From Table 2, we can see that severe misunderstanding exists for System 2 and further requirements work is needed. In fact, this observation was validated when, at the end of the actual system development, the user was not satisfied with the final product.

We also seeded the SRS with duplicate requirements to determine if such requirements could be identified using PFNETs. Table 3 shows the ccs based on path distances for each of the original and seeded requirements.

The ccs of the original and seeded requirements were very high when PFNETs for both groups were compared. Further analysis of the PFNETs for each group also revealed that the original and duplicate requirements were directly linked since they were perceived to be closely related. This provided us with evidence that PFNETs may be useful in uncovering duplicate requirements. In addition to the first four experiments, we ran a fifth similar classroom experiment the following academic year and achieved essentially the same results. Together, these five experimental results encouraged us to validate the PFNET approach in industrial settings.

Industrial Experimentation at Nortel

Our experiments were continued at Nortel, Inc., Dallas, Texas, [14] with their assistance over two semesters (about eight months). Our results again indicated that PFNETs were useful to identify misunderstood and duplicate requirements.

The procedure used to apply this technique was very similar to that outlined for our classroom experiments. A Web-based tool was introduced to collect the categorization data from each stakeholder. This tool interface consisted of check boxes to aid the process of categorization as shown in Figure 3 (see next page). Each Web page consisted of a requirement with the description provided at the top of the page. The remaining requirements were displayed on the same page. The stakeholder then checked the boxes of the requirements that

Software Systems	Overall Correlation Coefficient
System 1	0.77
System 2	0.46
System 3	0.91
System 4	0.87

Table 1: *Overall Correlation Coefficients Between Developers and User PFNETs*

Table 2: *Percentage of Requirements By Range of Correlation Coefficients*

Correlation Coefficient (cc) / Systems / Percentage of Requirements	cc >=0.9	cc < 0.9 and cc > = 0.7	cc < 0.7
System 1	43.75	21.88	34.38
System 2	0.00	0.00	100.00
System 3	50.00	50.00	0.00
System 4	50.00	50.00	0.00

Table 3: *Correlation Coefficients of Original and Seeded Duplicate Requirements*

Software Systems	Original Requirement	Seeded Requirement	Correlation Coefficient
System 1	1-13: E-mailing a Student Resumé to a Company	1-30: Sending a Student Resumé to a Company by E-mail	1.00
	1-16: Faxing a Student Resumé to a Company	1-30: Sending a Student Resumé to a Company by E-mail	0.96
	1-12: E-mailing a Student Transcript to a Company	1-31: Sending a Student Transcript to a Company by E-mail	1.00
	1-15: Faxing a Student Transcript to a Company	1-31: Sending a Student Transcript to a Company by E-mail	0.96
	1-11: E-mailing a Letter of Recommendation to a Company	1-32: Sending a Letter of Recommendation to a Company by E-mail	1.00
	1-14: Faxing a Letter of Recommendation to a Company	1-32: Sending a Letter of Recommendation to a Company by E-mail	0.96
System 2	2-2: Add Appointment	2-31: Make Appointment	0.93
	2-3: Change Appointment	2-32: Modify Appointment	1.00
	2-13: Delete Appointment	2-32: Modify Appointment	1.00
	2-20: Modify User	2-33: Add And Delete User	1.00
System 3	3-8: Add a Major	3-39: Modify a Major	0.98
	3-9: Edit a Major	3-39: Modify a Major	1.00
	3-10: Delete a Major	3-39: Modify a Major	0.98
	3-4: Add a College	3-40: Modify a College	1.00
	3-5: Edit a College	3-40: Modify a College	1.00
	3-6: Delete a College	3-40: Modify a College	1.00
System 4	4-23: Display Available Vehicle	4-30: Check Available Vehicle	1.00
	4-5: Delete Reservation	4-31: Cancel Reservation	1.00
	4-9: Make Reservation	4-32: Add Reservation	0.99

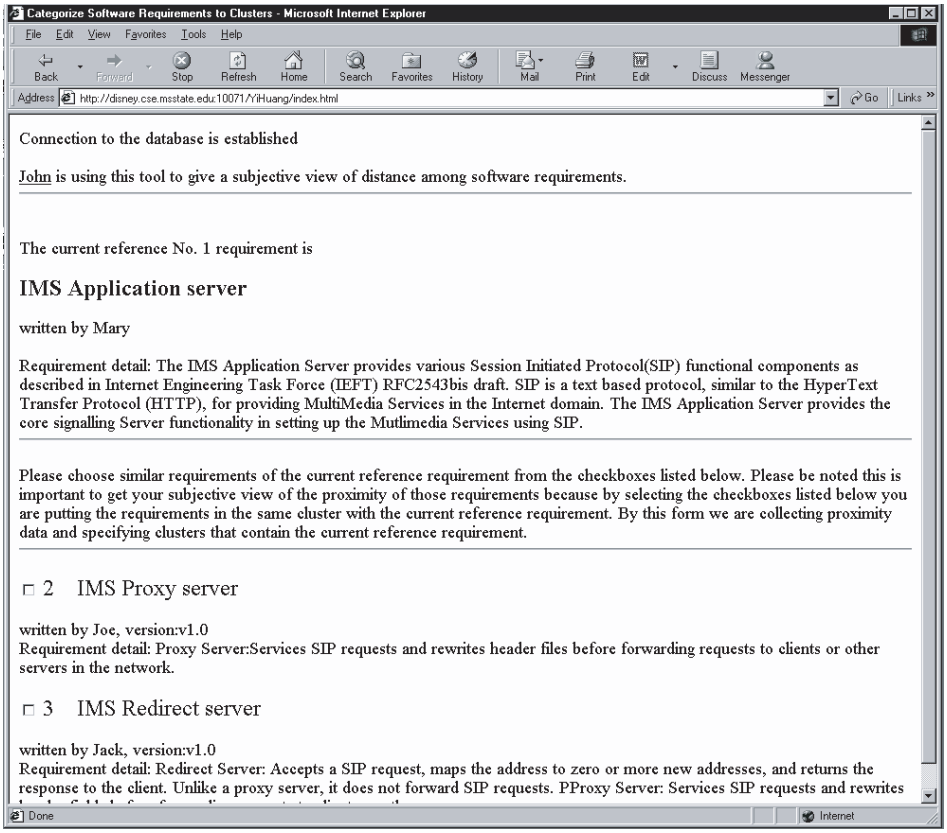


Figure 3: *Web-Based Check Box Interface at Nortel, Inc.*

were related to the requirement provided at the top of the page. After categorizing all the requirements, the information for each stakeholder was submitted over the Internet.

A comparison of customer and developer PFNETs revealed that two requirements (Nos. 6 and 14) had very low ccs, even though the overall cc between the customer and developer networks was 0.88. The shaded rows in Table 4 show the ccs for these two requirements. This demonstrates that the PFNETs in Table 4 were found to be duplicates after facilitating a session between the two groups. In this case, very high ccs (1.00 and 0.99) led to identifying this duplication. High values of correlation may also mean, however, that requirements might well be understood between the stakeholders. Only after further analyzing these requirements can we understand if duplication is present.

Table 4: *Correlation Coefficients of Customer and Developer of Individual Requirements for the Interactive Multi-Media Server (IMS)*

Requirements	Correlation Coefficients
1. IMS application server	0.95
2. IMS proxy server	0.87
3. IMS redirect server	0.99
4. IMS registrar server	0.94
5. IMS location server	1.00
6. IMS external interface	0.05
7. IMS hardware platform	1.00
8. IMS database interface	0.78
9. IMS SIP interface	0.67
10. IMS H.323 client interface	0.74
11. IMS PSTN gateway interface	0.67
12. IMS media gateway	0.79
13. IMS media server	1.00
14. IMS multi-domain support	0.05
15. IMS performance and capacity	1.00
16. IMS BBUA component	0.75
17. IMS application server security	1.00
18. IMS discriminator service	0.98
19. IMS arbitrator service	1.00
20. IMS network handling	0.69
21. IMS call transfer service	0.99
22. IMS call conference service	1.00
23. IMS accounting	0.77

Conclusion

Our initial research work, especially for the student projects at MSU and the project at Nortel, Inc. had generated encouraging results regarding the ability of the pathfinder technique to predict misunderstandings about requirements among customers and developers during the requirements analysis phase.

Further research conducted at AmerInd Inc., validated the usefulness of the technique in a typical industrial setting. The research also showed that this technique is scalable to a medium-scale project like New Material Acquisition. (A full version of this article, including the AmerInd Inc. and New Material Acquisition examples, appears on the CROSSTALK Web site at <www.stsc.hill.af.mil/crosstalk/2004/05/0405kudikyala.html>).

The ccs generated from the PFNETs of the stakeholders identify potentially misunderstood requirements. The values of ccs enable the groups to focus on potentially misunderstood requirements during facilitation sessions. Furthermore, duplicate and ambiguous requirements were also identified. Additional information concerning tools, processes, and experimental results are available from the authors.◆

Acknowledgements

The authors wish to acknowledge the support of the National Science Foundation (Grant #CCR-0303554); the James Worth Bagley College of Engineering at Mississippi State University; Nortel Inc. of Dallas, Texas; and AmerInd Inc. of Alexandria, Va. Without such support, this research and results would not have been possible. We are also grateful to the editorial staff of CROSSTALK for their patience and helpful suggestions to make this report better and more readable.

References

1. Keil, M., et al. "A Framework for Identifying Software Project Risks." Communications of the ACM 41.11 (1998): 76-83.
2. Davis, Alan, et al. "Identifying and Measuring Quality in a Software Requirement Specification." Software Requirements Engineering. Eds. R.H. Thayer and M. Dorfman. Los Alamitos: IEEE Computer Society Press, 1997: 164-75.
3. Boehm, B., and Victor R. Basili. "Software Defect Reduction Top 10 List." Computer 34.1 (2001): 135-37.
4. Faulk, S.R. "Software Requirements: A Tutorial in Software Requirements Engineering." Software Requirement Engineering. Eds. R.H. Thayer and M. Dorfman. Los Alamitos: IEEE Computer Society Press, 1997: 7-22.
5. Caroll, M.J., and J.R. Olson. "Mental Models in Human-Computer Interaction." Handbook of Human-Computer Interaction. Ed. M. Helander. North-Holland: Elsevier Science Publishers B.V., 1989: 45-60.
6. Faro, A., and D. Giordano. From User's Mental Models to Information System's Specification and Vice Versa by Extended Visual Notation. Proc. of the International Professional Communication Conference (IPCC '95), Savannah, GA., Sept. 1995.
7. Dearholt, D.W., and R.W. Schvaneveldt. "Properties of Pathfinder Networks." Pathfinder Associative Networks:

Studies in Knowledge Organization. Ed. R. Schvaneveldt. Norwood, N.J.: Ablex Publishing Corp., 1990: 1-30.
8. Cooke, N.M., and R.W. Schvaneveldt. "Effects of Computer Programming Experience on Network Representations of Abstract Programming Concepts." International Journal of Man-Machine Studies 29 (1988): 407-27.
9. Goldsmith, T.E., and P.J. Johnson. "A Structural Assessment of Classroom Learning." Pathfinder Associative Networks: Studies in Knowledge Organization. Ed. R. Schvaneveldt. Norwood, N.J.: Ablex Publishing Corp., 1990. 241-53.
10. Kudikyala, Udai K., and Rayford B. Vaughn Jr. "Software Requirements Understanding Using Pathfinder Networks: Discovering and Evaluating Mental Models." Journal of Systems and Software (to be published in 2004).
11. Kudikyala, Udai K., and Rayford B. Vaughn Jr. Software Requirements Understanding Using Pathfinder Networks as Mental Models. Proc. of the 2004 ASEE Southeastern Section Conference, Auburn, AL., Apr. 2004.
12. Lu, X. "Using Pathfinder Networks to Analyze and Categorize Software Requirements." Master's Thesis. Mississippi State University, 2000.
13. Goldsmith, T.E., and D.M. Davenport. "Assessing Structural Similarity of Graphs." Pathfinder Associative Networks: Studies in Knowledge Organization. Ed. R. Schvaneveldt. Norwood, N.J.: Ablex Publishing Corp., 1990: 75-87.
14. Yi, H. "Automated Web-Based Tool for Software Requirement Refinement Using Pathfinder Networks." Master's Project. Mississippi State University, 2001.

Note

1. AmerInd Inc., <www.amerind.com> is a medium-sized computer services company located in Alexandria, Va.

About the Authors

Udai K. Kudikyala is a Ph.D. candidate at Mississippi State University and an active research assistant. He has two years of teaching experience. His research interests include requirements engineering, especially modeling and evaluation of requirements understanding using artificial intelligence techniques, and network routing and parallel computing. Kudikyala received a Bachelor of Engineering in electronics and communication engineering from S.R.K.R. Engineering College, Andhra University, Bhimavaram, India, and a Master of Science in computer science from Mississippi State University.

Department of Computer Science and Engineering
Center for Computer Security Research
P.O. Box 9637
Mississippi State University
Mississippi State, MS 39762
Phone: (662) 325-7503
Fax: (662) 325-8997
E-mail: kumar@cse.msstate.edu

Rayford B. Vaughn Jr., Ph.D., is professor of computer science at Mississippi State University. A retired Army colonel, he served 26 years, including commanding the Army's largest software development organization and creating the Pentagon Single Agency Manager organization to centrally manage all Pentagon information technology support. After retiring from the Army, he was vice president of Integration Services, Electronic Data Systems Government Systems. Vaughn has more than 40 publications and actively contributes to software engineering and information security conferences and journals. He has a doctorate degree in computer science from Kansas State University.

Department of Computer Science and Engineering
Center for Computer Security Research
P.O. Box 9637
Mississippi State University
Mississippi State, MS 39762
Phone: (662) 325-7450
Fax: (662) 325-8997
E-mail: vaughn@cse.msstate.edu

Efficient and Effective Testing of Multiple COTS-Intensive Systems

Dr. Richard Bechtold
Abridge Technology

Thursday, 22 April 2004
Track 5: 1:30 - 2:15
Room: 150 G

Testing any complex system is a challenge, but testing multiple commercial off-the-shelf (COTS)-intensive systems is especially challenging due to a variety of factors. These factors include (1) systems combining highly diverse technology levels, (2) organizational users spanning numerous skill levels, (3) incorporating wide varieties of COTS vendors and packages, and (4) updated systems arriving for testing from projects that already may be substantially behind schedule and over budget. Regardless of these challenges, it is still essential to conduct efficient and effective testing in the context of limited time and nominal budgets. This article explains how to analyze, prioritize, plan, and manage the testing of COTS-intensive systems. Priorities are discussed from the perspective of identifying general types of transaction dialogues and specific instances of key dialogues. Dialogue value escalation is explained. Test planning and management are then discussed with the goal of maximizing the delivered benefit to organizational end-users.

Many organizations depend critically upon the maintenance and evolution of multiple in-house commercial off-the-shelf (COTS)-intensive systems. Most COTS systems are not matched perfectly to these organizations' specific needs and hence companies are often adjusting parameters, creating scripts, developing macros, writing code, or otherwise changing or extending functionality. Additionally, COTS vendors are always developing new packages, adding additional capability, and releasing updated versions of their products. Then, of course, there are the usual service packs, security patches, and other bug fixes that vendors want companies to install. And do not forget ongoing hardware and network upgrades. All this translates into a significant challenge for those responsible for performing testing to ensure adequate quality within this complex infrastructure.

For most organizations, testing everything is simply not an option. Hence, somehow testing must be biased in a manner that results in the maximum reduction of organizational risk, and that helps ensure the delivery of maximum benefit to system end-users. Amazingly, approximately half of all software modules in a system are defect-free and do not require any testing at all. Further, 80 percent of the defects come from 20 percent of the modules, and 90 percent of system downtime comes from at most 10 percent of the defects [1].

The potential exists to substantially improve upon system uptime and overall end-user benefit by finding, for example, just five percent more defects than are currently found. Ironically, end-user benefit can even be improved by finding *fewer* defects. This might occur if the testing approach is adjusted to focus on finding defects that cause the greatest system downtime. For example, the user experience will likely be substantially better if one defect is found that would crash the system daily versus finding three defects that would each crash the system annually.

Given the preceding data, a significant majority of testing efforts should be focused on 10 percent to 20 percent of any particular COTS system. The question is, how do we find that 20 percent? And since some in-house systems are clearly more critical than others, how do we account for that?

This article briefly identifies these and similar major challenges associated with performing efficient and effective testing of multiple in-house COTS-intensive systems. Then a nine-step process is explained that can help improve the approach to performing testing. The steps are as follows:

1. Identify Transaction Dialogues.
2. Analyze User Impact.
3. Determine Historical Defect Prevalence.
4. Prioritize Dialogues.
5. Prioritize Test Types.
6. Build Test-Dialogue Matrix.
7. Analyze Test Resources.
8. Establish a Test Plan.
9. Conduct Test Management.

Although the above sequence of actions may, at first, seem a bit daunting, several of the steps become substantially simpler after the first planning iteration. For example, once you have prioritized the types of tests to perform, you can typically reuse the same prioritization during future planning iterations. Additionally, this method is self-correcting. In earlier planning iterations, it is likely that you will have to rely upon a lot of guesswork to perform some of the steps. However, as you collect data during subsequent testing cycles, you can start using that data to replace, or at least augment, estimates. Note that the overall testing approach described in this article can also be applied to the testing of virtually any complex system, or for performing testing in environments where test resources are very scarce or test urgency is very high. However to simplify this discussion, the remainder of this article specifically focuses on discussion and examples relevant to testing multiple COTS-intensive systems.

COTS Testing Challenges

Testing of any complex software system can be challenging, and testing of COTS systems even more so because they require testing someone else's software (and hardware) and also testing your implementation and/or extension of that system. Deploy a few dozen COTS-intensive systems into the organization, and you have created a full-scale testing nightmare.

When performing testing in this type of environment, typically you need to consider and address most or all of the following issues:

- Different COTS systems may be based upon significantly different technologies.
- System end users usually have substantially different skill and experience levels.
- Your organization may depend upon a wide variety of COTS vendors.
- The threat levels presented by various COTS systems relative to core mission or business processes can differ by

orders of magnitude.

- System test work requests are not evenly distributed over time, and multiple parallel systems under test may rapidly overwhelm available test resources.
- Systems often become available for testing substantially later than originally planned.
- For some system failures, finding the actual source of the failure – or even re-creating the failure – can be quite complex and time-consuming, especially when the COTS system involves products from multiple vendors.
- The end-user organization may be reluctant to provide, or is incapable of providing (due to security reasons), actual production data for use in testing.
- Funds or resources originally allocated for system testing are often used to cover cost overruns during earlier project phases.
- System release or *go live* dates are often unmovable, or at least quite painful to move.
- Test organization resources are often redirected to perform unplanned emergency testing of systems to investigate problems reported by end users working with production systems.
- Calculating *return on investment* for software testing can be extremely difficult, or impossible [2, 3].

In summary, efficient and effective testing of multiple in-house systems requires a thorough understanding of the role those systems play and the value they provide – and the threat they present to the organization. It also requires a highly flexible plan to accommodate potentially significant organizational dynamics. The following nine-step process addresses these issues and is based upon these fundamental principles: (1) collect intelligence, (2) prioritize objectives, (3) understand resources, and (4) go for maximum impact.

1. Identify Transaction Dialogues

The COTS system value and impact analysis commences with identifying and documenting transaction dialogues. A transaction dialogue is a closely related set of actions – ideally mapped to one or more system requirements – that are normally performed at the same time, and that typically conclude at the same place they commenced. For example, adding new employees to the employee database can be viewed as a dialogue. It likely starts and ends on the same menu screen, and the data entry actions for adding employee information are closely related.

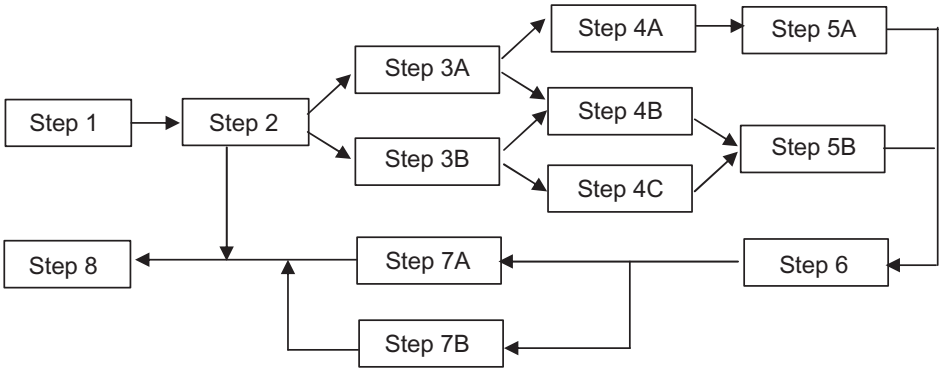


Figure 1: *Identifying and Diagramming Transaction Dialogues*

Another example of a dialogue is updating electronic timesheets with the activities performed and hours worked during a given day. Further examples include entering sales orders into an accounts receivable system, entering paycheck information into a payroll system, or paying invoices within an accounts payable system. When creating transaction dialogues, remember that the ultimate objective is to structure or design them so that they directly facilitate testing. (Note: If you already have a set of use-case scenarios [4], you can use those in lieu of creating transaction dialogues. In the absence of use-case scenarios, defining dialogues will likely be faster and easier than defining scenarios).

Dialogues can be easily diagrammed as a directed graph consisting of a set of nodes and arcs where the nodes represent specific (and possibly complex) activities or steps, each with a descriptive label and possibly a brief description. The arcs, each of which connects two nodes, indicate the sequence in which these activities can occur. As shown in Figure 1, multiple arcs departing from a node indicate that any of the destination actions might occur next. Similarly, multiple arcs arriving at a node indicate any of the source activities may have occurred previously. Note that there are no semantic differences between two arcs departing (as in Step 2) or in one arc departing and splitting (as in Step 6). Likewise there is no difference between multiple arcs arriving (as in Steps 4B and 5B) or multiple arcs joining and then arriving as one (as in Steps 6 and 8).

In order to ensure clarity and reduce misunderstanding, it is important to keep these drawings, and their associated semantics or rules, simple (i.e., these diagrams do not require the formality or detail of state transition diagrams). The objective of developing these diagrams is just to clearly indicate the boundary of the closely related set of system and user actions or steps that are covered by the dialogue.

It may be infeasible or unnecessary to try to capture all the various dialogues within a system. Hence, develop dialogues in a somewhat prioritized order. That is, what are the most important things a given system does? Capture that at a high level. Then, resources permitting, decompose complex nodes on higher-level dialogues into their own dialogues. If you are preparing to perform parallel testing of multiple COTS systems, then continue with this strategy by documenting the most important dialogues in the most important COTS systems, etc.

Again, resources and time permitting, steadily extend both coverage and formality by adding dialogues related to less-important systems and less-important system interactions, and by adding additional information to previously developed higher-priority dialogues. For the most important dialogues, consider augmenting them with comprehensive use-case scenarios.

When identifying dialogues within any given system, the ultimate objective is to gain a better understanding of where, how, and why users interact with that system. This sets the foundation for analyzing user impact.

2. Analyze User Impact

As you build a collection of transaction dialogues, you can begin to identify specific *user groups* within the end-user organization. Note that the groups you identify may not align with actual job titles held by the end-users. This is common and acceptable because your primary objective in this step is to improve your understanding of how closely related sets of people – regardless of their titles – interact with the various systems during day-to-day usage.

Depending upon the complexity of the end-user organization(s), you may also find it convenient to define a set of user-group types. Type distinction might be quite simple such as distinguishing between *beginning*, *intermediate*, and *advanced* users. For each identified user-group (or

group-type combination), assign a code to indicate that group's estimated contribution to core mission or business performance. For example, use A to indicate a group that is critical to core performance, B to indicate a group that is important, and C to indicate a nonessential support group.

Next, develop estimates of the number of users within each group. Typically the sum of this number across all user-groups will be much larger than the total number of users because some people will belong to multiple groups within one system. Additionally, numerous people or organizations may interact with multiple systems.

The last and possibly most difficult part of this step is to generally estimate the number of times an average user invokes a given dialogue. For example, an average user within a particular group might invoke one dialogue an average of once per month, and another dialogue 20 times per day. Ideally, you can collect feedback from system end-users to help you develop these estimates. Alternatively, you may need to resort to best-guess estimates developed by you, your test team, and any available systems analysts.

The final objective of this step is to have some idea regarding the relative frequency that various dialogues are occurring. This data will eventually be used to help prioritize dialogues, but that prioritization will also depend upon a dialogue's historical defect profile, so that step must be taken next.

3. Determine Historical Defect Prevalence

For each dialogue, document the average number of defects normally found during testing, and add the average number found after testing (that is, failures reported by system users). This is a relatively simple step; you will either have this data or you will not. Probably, you will not. Nevertheless, if you have been in the test group for a while you can likely invent some reasonably usable numbers. Alternatively, you may be able to use historical data from similar dialogues elsewhere within the system. These numbers do not need to be either accurate or precise. All we are trying to determine is which dialogues tend to contain the most defects. Whether a given dialogue has been 10 times more defective than another or 20 times more defective is not really important. For the prioritization step, it is sufficient to know that it is normally *much* more defect prone.

4. Prioritize Dialogues

Eventually, we are going to be testing based upon dialogues (and possibly use-case scenarios). The question at this point is which ones and how thoroughly? As indicated previously, the overall objective is to perform efficient and effective testing of multiple in-house COTS-intensive systems by striving to perform testing in a manner that reduces organizational risk and that helps ensure the delivery of maximum benefit to the end-users. Given this objective, relative importance can be established by examining the following for each dialogue:

- The size of the user-groups that invoke the dialogue.
- The frequency of invocation per unit time (e.g., 50 users, 20 invocations per user per day).
- The relative role of the user-group and dialogue to the organization's core mission capabilities.
- The historical defect prevalence associated with the dialogue.

Note that this step is an analytical step, not a calculation. The data developed in prior steps is used to help make highly informed decisions regarding how to prioritize dialogues. However, remember that much of the data may be gross estimates at best and some may be outright speculation, especially during early planning iterations. Hence, use the underlying numbers as an aid and combine them with the experience and judgment of you and your team.

The ideal output for this step is a lifeboat ranking (or a *fully ordered set*) of all the dialogues that are candidates for the next phase of testing. Basically, you start at the top of the list and keep working downward as far as you can go, resources permitting. If a fully ordered approach is infeasible in your environment, then strive to arrange dialogues into prioritized testing sets (e.g., *critical, highly important, important,* etc.).

5. Prioritize Test Types

This step consists of determining the types of tests to run relative to the dialogues. That is, are you only interested in testing features, or do you also consider other tests to be important? Examples of other types of tests include the following:

- Performance (normal load).
- Stress (approaching design limits).
- Overload (substantially beyond design limits).
- Security.
- Safety.

The purpose of this step is to ensure that everything that is important regarding organization risk management and end-user benefit is considered. For some environments the prioritization of which types of tests should be run first is fairly stable. For example, in life-rated systems, safety is typically the highest priority. After this step, you have everything necessary to build the test dialogue matrix.

6. Build a Test-Dialogue Matrix

This is another potentially simple step where you build a matrix to help ensure that all appropriate tests for each of the designated dialogues are run. For example, although you may be planning to conduct testing focused specifically on security, there may be numerous dialogues where security is not an issue and security testing is not required. Within this matrix (dialogues, in priority order, shown as rows; and test types, also in priority order, shown as columns), simply mark any cell to indicate that you want to perform that particular test (column) on that particular dialogue (row).

7. Analyze Test Resources

The last step to perform before building your test plan is to consider the resources available for testing. This includes not only human resources but also any automated tools you have that can be used to help with any subset of the testing. The general approach is to run the most important tests and to cover the maximum amount of highest priority dialogues. However, the overall objective continues to be risk reduction and user benefit, so it might make sense to adjust the order of the tests because you have one or more automated tools that allow you to rapidly and effectively test numerous lower priority dialogues. This would be the preferred approach when the compound benefit of testing a large number of lower priority dialogues exceeds that of the delayed higher-priority tests.

Integral to this approach is the fact that it is often difficult, if not impossible, to know your test resource availability relative to future demands that may be placed on your test group. This is why each of these steps is ultimately about prioritization and best utilization of available resources. For example, you may think you still have a month to finish testing the dozen COTS systems that are currently under test, but then again, what if you do not? Especially when it comes to the highly dynamic environments typically associated with testing multiple COTS systems, it is imperative that whatever is most important to occur next (whether it is a

single human-intensive test or a highly automated set of tests) is precisely what occurs.

8. Establish a Test Plan

Dynamic environment notwithstanding, it is important to take the time to develop an actual test plan, or to post your latest updates and revisions to the existing test plan. The formality and scope of various test plans may vary significantly depending on system criticality (or lack thereof), your overall test strategy, and resource availability. For example, there undoubtedly will be situations where exploratory testing [5] will give far better coverage (and test resource utilization) than scripted testing; additionally, you minimize the creation of comprehensive test documents to maximize the performance of actual test activities. Similarly, you will likely want to consider some level of combinatorial testing [6] to reduce the overall number of required tests by focusing on two-way and three-way parameter combinations for effectively finding multi-mode software faults. Regardless of the overall strategy, however, in the interests of communication, consensus, and overall risk management, some level of documented test plan is almost always beneficial.

In many environments, test plans (formal or otherwise) are frequently overtaken by events and therefore need frequent revision. For example, you may want to review your test plan on a weekly or monthly basis to determine if any revisions are required. Your test schedule, however, including revisions to dialogue prioritization, might need to be updated on a daily or weekly basis.

When planning, consider the various development groups within your organization, and absolutely consider their track record regarding their commitments to the test group. Do they normally deliver system and software enhancements on time, or do they routinely miss by a few months? The problem is, although you have to consider this when preparing your test schedule, project managers might find it a tad offensive if your schedule shows, for example, you are not really expecting them to deliver until six months after their claimed completion date. Therefore, consider keeping both an "official test schedule" and a "pessimistic test schedule." The pessimistic schedule is what you use to remind yourself how you think events are really going to unfold.

As you are building or revising your plan and schedule, always consider the possibility of obtaining additional test resources. For example, can developers help with system testing? Absolutely. Can system designers test? Certainly. Are systems engineers, requirements analysts, and end users all potentially effective testers? Of course they are. Are any of these resources available to you? They probably are not. But then again, maybe that depends on the relative importance of the items on the upcoming testing schedule relative to some of the other work currently occurring within your organization. Given the steps you have taken to create the inputs to your multi-system test plan and schedule, if you think additional resources are needed, at a minimum you can support your request with a very compelling rationale.

9. Conduct Test Management

Effective test management requires a wide variety of skills and activities, including the identification, collection, and analysis of a variety of test-related and quality-related metrics, and metrics associated with test status tracking, management, and control; proper reviews (to varying levels of formality) of test documentation and support material; and the determination of clear criteria for objectively assessing whether or not a system is ready for piloting, and when it is ready for operational use. Within this context, test activities should be prioritized with the ultimate objective of delivering maximum benefit to the end-users.

"The potential exists to substantially improve upon system uptime and overall end-user benefit by finding, for example, just five percent more defects than are currently found."

Prioritization is also influenced by the overall test strategy (such as exploratory testing and/or combinatorial testing). As testing activities are performed, there is normally a wealth of defect data collected. However, much of this strategically valuable data often is eventually lost. That is, test results are compiled and sent back to the development teams and are used for system and software corrections, but that data is not otherwise archived and managed by the test organization.

Note that one of the critical steps previously described is the determination or estimation of defect prevalence by dialogue. Since you are running the tests, you absolutely can and should retain this information. Ideally, you have (or can find or create) one or more failure-related reference lists so that system failures can be assigned to failure categories, failure severities, failure likelihood in the field, etc. This data can then be stored in a database, or even in a spreadsheet for use during future planning iterations.

Possibly even more important, to continue to enhance this overall process you must collect relevant data regarding failures or problems reported from system beta testing and production usage. These are the defects that slipped past your testing process. What are they? Where are they? When were they discovered? How were they discovered? For a given defect, which user group reported it first? Which user group reported it the most? What was the impact? This data is absolutely essential to you in your ongoing efforts to understand end-user consequences and organizational risk, and to continually improve the contributions of your test organization to overall mission success.

Delivering Maximum Benefit

Delivering maximum benefit to organizational end-users, and ultimately to the organization, is everyone's job. Therefore, related to this discussion are numerous quality topics such as the value of peer reviews, the benefits of early inspections, the greater benefits of perspective-based reviews [1], etc., which directly impact overall organizational competence and mission performance. However, as a manager or technical specialist within the test organization, you normally do not influence such things. The projects build what they build, the procurement organization buys what it buys, and the maintenance teams implement changes however they feel like. And you are the last line of defense, literally, for your organizational system's end-users.

There are many alternative approaches to testing COTS-intensive systems. You can, for example, assign functional requirements to different priority levels, and then take a priority-driven approach to test performance. Alternatively, many organizations, if they have sufficient resources, plan and perform tests by taking one of these approaches: *try-to-test-everything-once*, *top-down*, *front-to-back,* or any of numerous other techniques.

The premise behind the approach described in this article is to increase your

efforts relative to test priority analysis and planning, and to strive to maximize the test organization's contribution to the overall delivery of benefits to systems' end-users, and the overall end-user experience. Invariably, testing priorities will continue to be a constantly shifting landscape due to unexpected events. However, by regularly following the steps described in this paper you will steadily improve your understanding of user-group interactions with various systems and steadily increase the amount of data available to you related to user-discovered defects and consequences. This combination of strategy, understanding, and data will directly support and enhance your overall efforts toward leveraging your test organization to deliver maximum benefit to your end-users.◆

About the Author

Richard Bechtold, Ph.D., is president of Abridge Technology and is an independent consultant who assists industry and government with organizational change and systematic process improvement, especially in the area of implementing effective project management. Bechtold has more than 25 years of experience in the design, development, management, and improvement of complex software systems, architectures, processes, and environments. This experience includes all aspects of organizational change management, process appraisals, process definition and modeling, workflow design and implementation, and managerial and technical training. Bechtold is an instructor at George Mason University where he teaches graduate-level courses in software project management, systems analysis and design, principles of computer architectures, and object-oriented java programming. The second edition of his latest book, "Essentials of Software Project Management," is scheduled for publication in 2004.

Abridge Technology
42786 Oatyer CT
Ashburn, VA 20148-5000
E-mail: rbechtold@rbechtold.com

References

1. Boehm, B., and V. Basili. "Software Defect Reduction Top-10 List." NSF Center for Empirically Based Software Engineering, Jan. 2001 <www.cebase.org/www/home/index.htm>.
2. Bechtold, R. "Software Quality Valuation: Return on Investment Versus Reduction of Risk." International Conference on Practical Software Quality Techniques/Practical Software Testing Techniques. Washington, D.C., June 2003.
3. Bechtold, R. "Return on Investment for Software Testing." Software Testing Forum. Reston, VA., Mar. 2003.
4. Satzinger, J., and T. Orvik. The Object Oriented Approach, Concepts, System Development, and Modeling with UML. 2nd ed. Course Technology, Thomson Learning, 2001.
5. Bach, J. "What Is Exploratory Testing?" <www.satisfice.com/articles/what_is_et.htm>.
6. Daich, Gregory T. "No-Cost Combinatorial Testing Support." Software Technology Support Center, Hill AFB, UT., <www.stsc.hill.af.mil/consulting/sw_testing/improvement/cst.html>.

WEB SITES

Object Management Group

www.omg.org

Founded in 1989 by 11 companies, the Object Management Group (OMG) now has about 800 members. It is a not-for-profit corporation formed to create a component-based software marketplace by accelerating the introduction of standardized object software. The OMG is establishing the Model Driven Architecture through its worldwide standard specifications, including Object Services, Internet Facilities, Domain Interface specifications, and more.

The Agile Alliance

www.agilealliance.com/home

The Agile Alliance is a non-profit organization dedicated to promoting the concepts of agile software development, and helping organizations adopt those concepts. The site features an extensive library of articles about agile processes and agile development.

Center for Software Engineering

http://sunset.usc.edu/index.html

Dr. Barry Boehm founded the Center for Software Engineering (CSE) in 1993. It provides an environment for research and teaching large-scale software design and development processes, generic and domain-specific software architectures, software engineering tools and environments, cooperative system design, and the economics of software engineering. One of CSE's main goals is to research and develop software technologies that can help reduce cost, customize designs, and improve design quality by doing concurrent software and systems engineering.

INCOSE

www.incose.org

The International Council on Systems Engineering (INCOSE) was formed to develop, nurture, and enhance the interdisciplinary approach and means to enable the realization of successful systems. INCOSE works with industry, academia, and government to disseminate systems engineering knowledge, promote collaboration in systems engineering, establish integrity in systems engineering standards, and encourage research and educational support to improve the systems engineering process and its practices.

Risk Management

www.acq.osd.mil/io/se/risk_management/index.htm

This is the Department of Defense (DoD) risk management Web site. The Systems Engineering group within the Interoperability organization formed a working group of representatives from the services and other DoD agencies involved in systems acquisition to assist in the evaluation of the DoD's approach to risk management including the latest tools and advice on managing risk.

Risk Factor: Confronting the Risks That Impact Software Project Success

Theron R. Leishman
Software Technology Support Center/Northrop Grumman

Dr. David A. Cook
The AEgis Technologies Group, Inc.

Wednesday, 21 April 2004
Track 5: 2:25 - 3:10
Room: 150 G

Every systems and software project involves risk. Often, how you manage your program risks is a deciding factor in the eventual success or failure of your program. If you ignore the risks, your program has a higher chance of failing. On the other hand, if you try to track and manage all possible risks, you can expend your entire budget managing the risks, and never produce a deliverable. Risk management, like any other element of systems and software development, requires forethought and careful planning. This article explains what risk management is, and then discusses some common developmental risks.

There is a popular television program that appeals to the brave of heart called "Fear Factor." During this program, contestants compete against each other in events and activities that cause considerable fear for most people. Although it is considered reality TV, the things the contestants are required to eat, drink, touch, dive into, or navigate through are not considered reality by the average person. Contestants are eliminated with each fear-defying competition, until a single winner remains.

In the *reality world* of software development and/or acquisition, there are circumstances, events, activities, etc. that instill fear in the hearts of software developers, acquirers, and managers. Too often these fear factors are ignored and have devastating impacts on software projects. The authors of this article have participated in the review and assessment of numerous software-intensive systems from the perspective of both developing software and purchasing software. From our experiences, we have identified several issues that are common to many software-intensive programs. These issues, *risk factors*, are keys to either the success or failure of any software-intensive system.

What Is Software Risk Management?

Risk is defined as, "A possible future event that, if it occurs, will lead to an undesirable outcome" [1]. In the context of software then, software risk management would logically imply the managing of possible future events that could have undesirable effects on software projects. That sounds simple enough. What future events could possibly negatively impact my software project? Risk management is simply defined as a generalized process for managing risks [2]. However, to be an effective risk management process, it has to be both accurate and usable; that is, it must provide results to a manager in time to allow the manager to make informed decisions that allow risks to be minimized or avoided.

Every development project has risks. The risks can range from the common, "We might not be able to find a JOVIAL programming language expert by next month," to the uncommon, "A hurricane might destroy all of the prototype aircraft." How you approach these risks is what is important. In the rest of this article, *risk* refers to software risk.

The Two Extremes of Software Risk Management

There are two extremes of software risk management: too little and too much. Each one can be explained in terms of popular operating systems.

The first extreme, the too little school, is much like older versions of UNIX. While UNIX is a fine operating system, its roots show a lack of disciplined software development. For those accustomed to a single version of an operating system, there existed numerous versions of UNIX – well over 100[1].

Here is an example of the *too little* school. One of the authors used to work on a Burroughs mini-computer back in the mid-80s. It ran a version of UNIX referred to as BNIX, which has an interesting property. The memory table that kept a list of all currently running jobs (the *process table*) had a limited size, but there was no mechanism to protect it. The size was somewhere around 1,023 entries. If you tried to run a process after the table was full, BNIX just *bumped* the counter to 1,024 and started entering process data.

The fact that the process table was full did not deter it – it just wrote to whatever memory was at the 1,024th position. Since the process table was stored in memory, it was likely (in fact, it was certain) that this 1,024th process just overwrote something critical in the operating system. Assuming that the system did not crash immediately, you could start a 1,025th process. Eventually, the operating system would crash in some bizarre fashion.

By not having mechanisms to deal with the problem, which was a limited size process table, BNIX used the *ostrich-head-in-the-sand* method of risk management. This method means you assume that all will go correctly, and you just pretend that no risks exist. Many current software development projects use this method – the equivalent of believing in magic, which often occurs because of a new and untested silver bullet. For example, "Now that we are using commercial off-the-shelf software in our program, we're going to cut the schedule by 25 percent." This is unrealistic, unproven, and head-in-the-sand thinking.

The second extreme of software risk management, the *too much* school, is more like Windows 98. Windows has many mechanisms to ensure you do not overflow process tables, overwrite system memory, or use memory that is currently allocated to other programs. However, Windows 98 sometimes gave a message that said something like, "Unable to run program – not enough memory. Try quitting a running program and try again." Unfortunately, this error message occurred when only running a single program, or even no program. What typically happened was that a previously running program had used memory incorrectly (or crashed prior to *cleaning up* after itself) and left the system memory corrupted. Windows 98 was unable to ensure that a new program would not be overwriting memory still marked as *in use*, and would not let the new program run. It usually required a reboot to set things right. This method, being *over-cautious* of the risks and always assuming that the *worst* is going to happen, is the other extreme.

To effectively manage risks, you have to

take the middle ground. You cannot ignore risks and pretend that all will go well. However, you cannot micromanage all possible risks; you do not have the time and the resources. To take the middle ground, you have to be aware of some common risk factors.

Critical Systems and Software Risk Factors

To adequately manage risks, it is essential to evaluate the program's/project's unique situation. There are, however, certain risks that tend to impact many programs/projects. Having evaluated several programs, we have identified the following critical risk factors as issues that impact the success of many programs. By addressing the key risk factors, programs can make great strides in managing risks that may impact them.

Inadequate Planning

Recently while assisting in the review of a large Department of Defense (DoD) software acquisition program, we asked to review the program's planning documents. In response to the request, the program office produced a Microsoft Project schedule. This response is common when we ask about planning documents while performing program reviews.

There appears to be a lack of desire, interest, ability, and attention to performing adequate program/project planning. Not all programs/projects require the same level of planning; however, they all should include enough planning to adequately assess their issues. Program/project plans should consider the following:

- Date and status of the plan.
- Scope of the plan.
- Issuing organization.
- References.
- Approval authority.
- Assumptions made in developing the plan.
- Planned activities and tasks.
- Policies, etc., that dictate the need for this plan.
- Micro-references – other plans or tasks referenced by the plan (other plans or task descriptions that elaborate details of this plan).
- Schedules.
- Estimates.
- Communication chains.
- Resources and their allocation.
- Responsibilities and authority.
- Risks.
- Quality control measures.
- Cost.
- Stakeholder identification.
- Interfaces among stakeholders.
- Environment/infrastructure, including safety needs.
- Training.
- Glossary.
- Change procedures and history.
- Schematics, diagrams, and architectures to further clarify the intent of the plan.

A lack of adequate planning tends to indicate a lack of forethought and direction. Inadequate planning at either a program or project level greatly increases risk to successful project completion. Elaine M. Hall [2] has a good explanation of how to plan for risk.

Unrealistic Schedules

The daughter and son-in-law of one of the authors of this article recently built a house. The couple was anxious to have the home completed so they could move in early enough in the fall to complete the yard before winter. To them this seemed a reasonable request.

As the couple negotiated the deal with the chosen contractor, they were told the house would not be completed in the time schedule desired by the couple. The contractor showed them the list of tasks to be completed and the dependency of each task with other tasks. He further explained to them the dependency on outside factors such as weather, availability of special materials that were part of the couple's design, availability of subcontractors, whims and schedules of building inspectors, and the priority of their house in relationship to other homes the developer was building. The bottom line: the couple was not able to eliminate critical steps in the building process, shortcut the availability of critical resources, or circumvent required inspections.

Have you ever been coerced into or managed a software project that begins with a predetermined schedule? One that is unachievable, to start with, that requires sound processes to be altered or eliminated? One that ignores the requirement for critical resources, eliminates critical reviews, and allows only minimal time for testing? If so, then in talk-show host and author Dr. Phil's words, *"What are you thinking?"*

Why do we agree to and/or impose software schedules that require the abandonment of everything that we have learned over the past several years as being essential to the success of software development or acquisition projects?

The authors have seen numerous software projects with schedules we consider to be unreasonable, in our opinion, due to the following:

- Schedules based on product need rather than a realistic assessment of engineering effort. This includes schedules based on an arbitrarily imposed due date.
- Effort estimates and resulting schedules based on hallucinations (or unrealistic hope) rather than on historical basis.
- Schedules based on unrealistic, incorrect, or unknown requirements.
- Schedules developed without adequate understanding of sound software engineering practices.

For these reasons, we have included unrealistic schedules as one of our software risk factors.

Unfortunately, managers who attempt to transform unreasonable schedules into accurate and reasonable ones run the additional risk of extreme upper management displeasure. This (and the risk of losing one's job) creates great pressure on lower-level managers to continue to pretend that unreasonable schedules are, in fact, achievable.

Unconstrained Requirements Growth

There have been volumes written over the years about the impact of incomplete, inaccurate, growing, unstable, and on, and on, and on, software requirements problems. In the April 2002 CROSSTALK, we enumerated several requirements risks that can drown software projects [3].

Research conducted by Capers Jones [4] identified the top five risks that threaten the success of software projects in various sectors. Figure 1 summarizes the approximate percentage of projects that suffer from creeping user requirements.

As indicated in Figure 1, the majority of management information systems and military projects suffer from requirements creep. We further saw that nearly half of the outsourced projects included in the study also suffered from requirements creep.

Our experience shows that nearly all projects suffer from some form of requirements risk. The impact is often catastrophic to the success of the project. We see requirements issues as a substantial risk to the success of many software projects.

Dysfunctional Organizational Culture

The following is the parable of the Happy and Productive Worker:

> *Once upon a time in a company not far away, there was a worker. He was a productive, happy worker, but alas, he was unsupervised.*
>
> *The company saw that the worker was unsupervised and made a supervisor.*
>
> *The coordinator saw that the worker was productive and happy and made a lead worker to make the worker more productive and happier.*
>
> *The company saw that the department in which the productive, happy worker worked*

had grown and made a manager to manage the department in which the productive, happy worker worked productively and happily.

The manager saw that the worker was productive and happy and made an assistant manager to help manage the department in which the productive, happy worker worked.

The company saw that the department where the happy, productive worker worked had grown and made an administrator to administer the department in which the productive, happy worker worked.

The administrator saw that the worker was productive and happy and made an assistant administrator to assist in administering the department in which the productive, happy worker worked.

More people were added until the director saw that the department was losing money. So he consulted a consultant. The consultant examined the department in which the productive, happy worker worked productively and happily and advised there were too many people in the department in which the productive, happy worker worked.

And the director paid heed to the counsel of the consultant and fired the productive, happy worker.

In the parable of the Happy and Productive Worker, the organization valued and rewarded the wrong things. As you think of the organizations you work in, what is the culture that exists? Is it a culture plagued with high turnover? Do you enjoy going to work each day, or does it take an act of God to get you out of bed and into the office? Do you have the training and skills required to perform the tasks that are expected of you? Are meetings useful with something actually being accomplished, or are there too many long, useless meetings that just distract from what you are trying to get done?

The organizational culture of many software companies appears to be that of working harder to get out of doing work rather than actually getting things done. This risk factor ties closely to the management. There is an inherent risk with over-managing – and perhaps this risk is greater than under-managing. Over-management forces workers to spend too much time justifying what they do (and do not do). Workers need to be allowed to fail occasionally, and learn from their failure. If you create an environment of *one mistake and you're out*, workers will spend so much time trying not to fail that they will not have the time to succeed.

During a recent seminar, we asked the attendees how many of them had ever been part of what they considered to be a high-performance team. Of 30 attendees, five indicated that they had. Upon discussion with the group, we identified the following characteristics of a high-performance team:

- Value for all team members.
- Open communication.
- Common understanding of team goals.
- Recognition for valuable contributions.
- Clear understanding of job expectations.
- Shared desire to meet or exceed expectations.
- Demonstrated commitment to the team by management.

It is difficult to successfully develop software in a dysfunctional organization. Management must look honestly at the organization to determine if it is dysfunctional. If your culture is one similar to that of the Happy Productive Worker, that culture is a serious risk to your organization's success in consistently providing quality software.

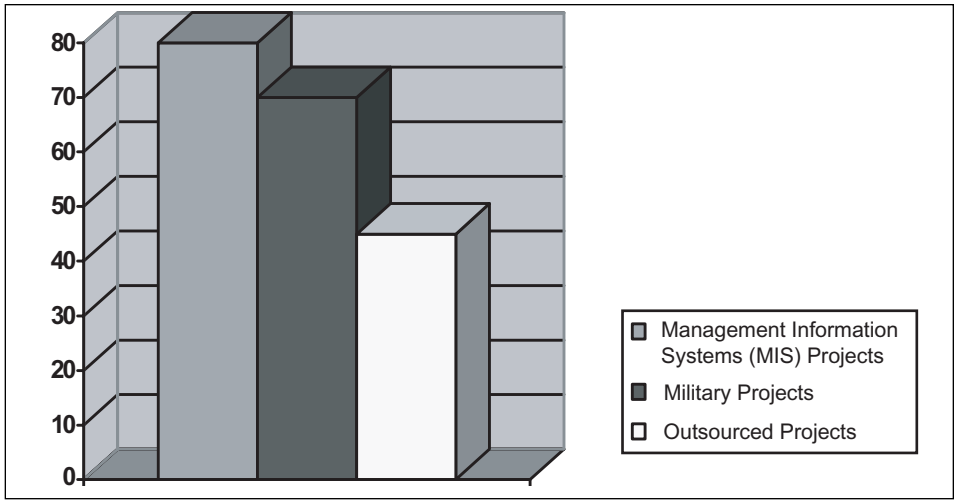


Figure 1: *Percentage of Three Types of Projects That Suffer From Requirements Creep*

Not Having or Following Processes

Having and following a good process is essential to the consistent development of quality software systems. It has been said that a software product is only as good as the process used to develop and maintain it.

There seems to be common agreement on the value of having and following sound processes in the development and maintenance of software, yet in the authors' experiences, this continues to be a weakness of many software development organizations. We see many organizations that either lack processes or do not follow their existing processes. Some have processes in place but are quick to abandon them when hit with schedule pressure.

When this happens, the good, smart things that help ensure quality software fall by the wayside and programs/projects gradually spin out of control. Not having or following processes is on our list of software risks because this is an area that is often only given lip service. If processes exist, and are required to be followed, many other risks immediately become less critical.

Failure to Actively Manage Software Risk

Have you ever thought about how much we depend on software? The world has evolved to the point where almost everything we do involves software. Lt. Gen. Jim Fain (U.S. Air Force retired), during his tenure as F-22 program director, described software's importance by saying, "The only thing you can do with an F-22 that does not require software is to take a picture of it" [5]. Now in 2004, there is probably not even a non-disposable camera in use that is not at least partially operated by software.

Software has made great technological advancement possible. This dependence on software has also brought with it consequences. Software failures have resulted in significant financial losses and even the loss of life. Software has become the very heart of the new economy, and business risk management must include software risk management to survive.

During a recent program review, one of the authors of this article asked how software risks were managed on the program. The program office produced a list from their risk database of software risks that included risks at a very high level and were generic to the point that they could be applied to any program. When asked how the list was generated, it was explained that a tool had generated the risks based on information provided. We call this risk management in a can. This was a large DoD program that included software with potential life-threatening consequences.

Our dependence on software has pushed us to the point where a proactive software risk management process is essential. Managers must determine whether any

unwanted events may occur during the development or maintenance of software, and make appropriate plans to avoid or minimize the impact of these negative consequences. Failure to do so may have devastating consequences. A good software risk management process should include the following steps:

- Identification of program-/project-specific software-related risks.
- Detailed analysis of each software risk.
- Development of plans to address software risks.
- Active monitoring and tracking of software risks.
- Use of metrics to monitor the risk-management process.

By actively managing software-related risks, the probability of experiencing first-hand the consequences of a software failure can be greatly reduced. While failure to actively manage software risk is the last risk mentioned, it is critical.

How detailed should your risk management be? On one recently performed assessment, the *risk list* on the project ran more than 400 pages, and had a complex formula showing each risk status. Such a complex risk list is almost useless. It is difficult to see the overall project risk status, and it takes a huge amount of time to keep it current.

The authors recently performed another assessment, and saw the risks briefed as a 25-item list, each with either a red, yellow, or green light (standing for high, medium, and low risk). This method was easy to use, easy to understand, and reasonably easy to update. The key point is that the program manager, using this simple 25-item list, has the information to understand the risk exposure of the program and make good decisions.

Conclusion

This article is not intended to present a complete list of risk factors for a particular program/project – it would take much more space. Likewise, it is not intended to be a primer on correct risk-management practices. What this article is intended to do is heighten your awareness of risk management, and give you a starting point in creating an appropriate risk management strategy. Risk management is usually a task best done by somebody who has some experience in the area. Locate an appropriate source of training or experience, and learn from other's mistakes. If you proactively manage risks properly, then you will spend little time reactively *putting out unexpected fires*, thus freeing up more time to build your system based on current, accurate, and easy-to-understand risk information.

During a recent "Fear Factor" episode, contestants were required to leap from one boat to another while the two boats were speeding next to each other. This resulted in some of the contestants experiencing very traumatic falls into the water.

In this article, we have identified six risk factors that, based upon our experience, can have significant impact on the success or failure of software programs and projects. By confronting these risks, many inherent problems can be avoided. We recommend the following:

- Take program/project planning seriously.
- Do not lie to yourself; develop schedules based on sound facts and proven approaches.
- Stringently control requirements growth.
- Establish a success-oriented organizational culture.
- Develop and follow sound software development, maintenance, and program/project management processes.
- Actively identify and manage risks specific to your program/project.

By doing these, you can reduce the probability of your program or project having a traumatic fall into the pit of unsuccessful software programs and projects. *The biggest risk of all is failing to manage your risks!* ◆

References

1. Leishman, T., and J. VanBuren. "The Risk of Not Being Risk Conscious: Software Risk-Management Basics." STSC Seminar Series, Hill AFB, UT., 2003.
2. Hall, Elaine M. Managing Risk. Addison-Wesley, 1998.
3. Leishman, Theron, and Dr. David A. Cook. "Requirements Risks Can Drown Software Projects." CROSSTALK 15.4 (Apr. 2002): 4-8 <www.stsc.hill.af.mil/crosstalk/2002/04/leishman.html>.
4. Jones, Capers. Assessment and Control of Software Risks. Prentice Hall, 1994.
5. Naval Postgraduate School. "Importance of Software to the Military." U.S. Navy, 4 Mar. 2000 <www.nps.navy.mil/wings/acq_topics/AcqTopics.htm>.

Note

1. See <www.scs.org/confernc/astc/astc04/cfp/astc04.htm> for a starting point.

About the Authors

Theron R. Leishman is a consultant currently under contract with the Software Technology Support Center at Hill Air Force Base, Utah. Leishman has 19 years experience in various aspects of software development. He has successfully managed software projects and performed consulting services for the Department of Defense, aerospace, manufacturing, health care, higher education, and other industries. He is a Level 2 Certified International Configuration Manager by the International Society of Configuration Management, and is employed by Northrop Grumman. Leishman has a master's degree in business administration from the University of Phoenix.

Software Technology Support Center
6022 Fir AVE, BLDG 1238
Hill AFB, UT 84056
Phone: (801) 775-5738
Fax: (801) 777-8069
E-mail: theron.leishman@hill.af.mil

David A. Cook, Ph.D., is a senior research scientist at AEgis Technologies Group, Inc., working as a verification, validation, and accreditation agent in the modelling and simulations area. He is currently supporting the Airborne Laser System Program. Cook has more than 30 years experience in software development and management. He was formerly an associate professor at the U.S. Air Force Academy, a deputy department head of the Software Professional Development Program at the Air Force Institute of Technology, and a consultant for the Software Technology Support Center. Cook has published numerous articles on software-related topics. He has a doctorate degree in computer science from Texas A&M University.

AEgis Technologies Group, Inc.
6565 Americas PKWY NE
STE 975
Albuquerque, NM 87110
Phone: (505) 881-1003
Fax: (505) 881-5003
E-mail: dcook@aegistg.com

The ~~Art~~ Technology of War

I entered the defense industry in the midst of the Cold War. The only action my systems saw was operational test and evaluation or an occasional Top Gun or Gunsmoke competition. It was a time of scientific tinkering and military conjecture.

On the conjecture side of the equation, then as is now, a 2,500-year-old Chinese general known as Sun Tzu wielded tremendous influence over American military strategy. In fact, his manuscript "The Art of War" may be the most quoted document in military history. When asked to estimate enemy forces, Army Gen. Tommy Franks replied, "The answer I'm going to give you will not be a number, because, as has been the case since Sun Tzu said it, precise knowledge of self and precise knowledge of the threat leads to victory."

While interesting reading for generals and corporate ladder climbers, most engineers find Sun Tzu too ethereal, more reminiscent of a Cheech and Chong elucidation than an Orwellian dissertation. In fact, the title turns many engineers away because art is a dirty word, unless you are a sanguine software tyro. Leave the art of war to Norman and Tommy while we examine the technology of war.

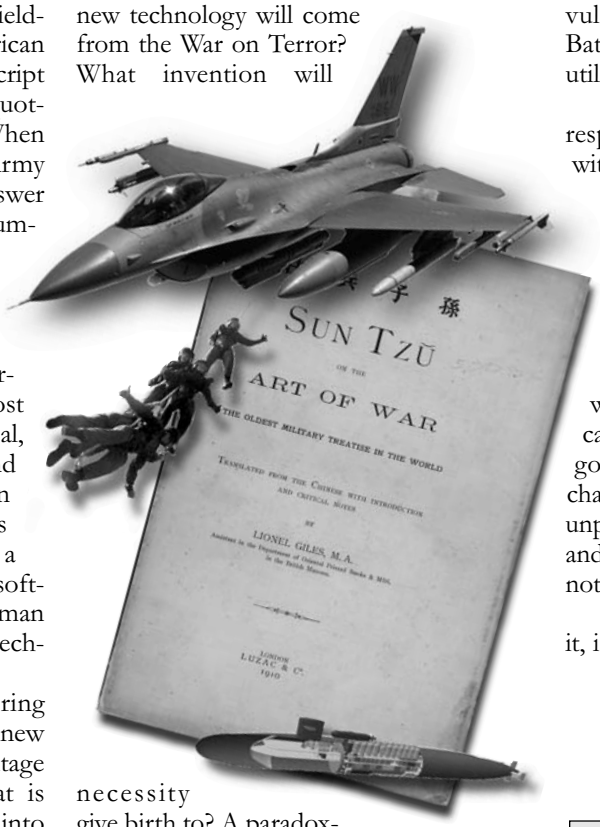
Wars give birth to society-altering technology. During conflict, radically new technologies surface to gain an advantage and quell one's adversary. Once that is accomplished, the technology pops into the commercial sector and alters the direction of humanity.

Eleven years after the Wright brothers flew at Kitty Hawk, World War I erupted. Sparsely used airplanes became courier, reconnaissance, and fighting machines, launching an aviation industry that abridged the world. That war also led to the development of radio. KDKA's historic radio broadcast in 1920 paved the way for Elvis, John, Paul, George, Ringo, Elton, Howard, Rush, and catchy advertising jingles.

World War II led to atomic fission and microwave communication, changing the way we pop popcorn forever. The first computer calculated artillery trajectories so one could accurately rain shells from afar. Now computers facilitate message trajectories so one can arbitrarily rain e-mail from afar and annoy with SPAM, another byproduct of World War II, the edible kind that is.

The Cold War launched satellites to spy. Now we spy with satellite dishes, perusing television's ersatz reality of survivors, bachelors, and apprentices. Now the Cold War is history, with Google at ground zero, and the BLOG intercontinental.

So what is next? What new technology will come from the War on Terror? What invention will necessity give birth to? A paradoxical question for sure.



Maybe Sun Tzu's tome can shed some light on our uncertainty. "Rapidity is the essence of war," he opines. "The pinnacle of military deployment approaches the formless. Then even the deepest spy cannot discern it or wise make plan against it." How does the largest military in the world approach rapid formless deployment? No one, short of George Lucas or Gene Roddenberry, can accurately predict, but here are some good bets.

Remote sensing and surveillance technology that can find, identify, and lock onto individuals versus structures. Smart bombs that get personal, if you will, reversing the situation and striking terror back into the terrorist. Can you hear me now? Good.

Faster processors coupled with intelligent and efficient software sort and match torrents of information from all over the world, finding patterns that predict attacks. That is just to reign in the Bernard Ebbers, Ken Lays, and Martha Stewarts of the world; imagine how it will thwart terrorists.

Bio and nanotechnology can increase a warrior's power while lightening his load. Decentralizing computing and weaving it into the fabric of the warrior increases flexibility and reduces predictability and vulnerability. This is the return of Batman's utility belt, complemented by a utility shirt, jacket, and pants.

These technologies enable rapid response, flexibility, and interoperability without interdependence. All key in a rapid formless deployment. Underlying it all is software. Unfortunately, software's unpredictability and cost are also the long pole in the tent of innovation, toiling to catch up with its hardware, bio, and nanotechnology counterparts.

Several decades ago, we turned to software for its flexibility. Early simple applications were easy to change, and beat going back to the fabrication shop to change the hardware. Increased complexity, unpredictable delivery times, low quality, and cost overruns indicate that software is not soft but rather hard.

Your challenge, if you choose to accept it, is to put the soft back in software.

— Gary Petersen
Shim Enterprise, Inc.
garyp@shiminc.com

Can You BACKTALK?

Here is your chance to make your point, even if it is a bit tongue-in-cheek, without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CROSSTALK, we also accept articles for the BACKTALK column. BACKTALK articles should provide a concise, clever, humorous, and insightful article on the software engineering profession or industry or a portion of it. Your BACKTALK article should be entertaining and clever or original in concept, design, or delivery. The length should not exceed 750 words.

For a complete author's packet detailing how to submit your BACKTALK article, visit our Web site at <www.stsc.hill.af.mil>.

Published by the Software Technology Support Center (STSC)

CrossTalk / MASE
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737