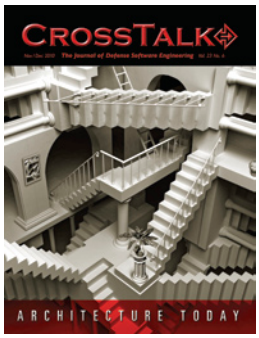
CROSSTALK

Nov / Dec 2010 The Journal of Defense Software Engineering Vol. 23 No. 6

ARCHITECTURE TODAY

Departments



Cover Design by Kent Bingham

Architecture Today

CROSSTALK

OSD (AT&L) Stephen P. Welby
NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Kasey Thompson
Article Coordinator Lynne Wade
Managing Director Brent Baxter
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

Phone 801-775-5555
E-mail stsc.customerservice@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Office of the Secretary of Defense (OSD) Acquisition, Technology and Logistics (AT&L); U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Security (DHS). OSD (AT&L) co-sponsor: Software Engineering and System Assurance. USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cybersecurity Division in the National Protection and Programs Directorate.

The USAF Software Technology Support Center (STSC) is the publisher of **CROSSTALK** providing both editorial oversight and technical review of the journal. **CROSSTALK'S** mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

Subscriptions: Visit <www.crosstalkonline.org/subscribe> to receive an e-mail notification when each new issue is published online or to subscribe to an RSS notification feed.

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.crosstalkonline.org/submission-guidelines>. **CROSSTALK** does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with **CROSSTALK.**

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of **CROSSTALK** are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CROSSTALK Online Services:
See <www.crosstalkonline.org>, call (801) 777-0857 or e-mail<stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

CrossTalk would like to thank the 309 SMXG for sponsoring this issue.

Kasey Thompson
Publisher

Architecture Today

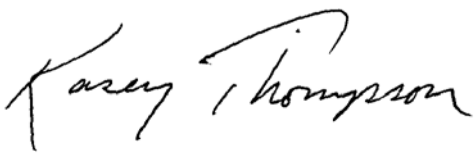
I find it comforting to know that some structure exists in a world of increasing confusion, fragmentation, and uncertainty. Amongst the crashing economies, warring nations, and natural disasters, we find software architectures quietly, resolutely, even purposefully adding stability to our software world which can be chaotic itself at times. While I may be waxing a little too poetic about software architecture in the face of larger calamities, you have to admit, there is certain serenity and solace in the construction and formation of peacefully co-existing relationships between systems. Call me a geek, but I get excited thinking about a well designed architecture which provides strength, security, foundation, functionality, and communication for the system users. Think of a world where an engineered substructure is designed and tested for all future users prior to adding additional systems and functions. Governments of the world take note; architecture adds to strength, durability, and potency when performed thoughtfully.

So where is Architecture Today? Grady Booch answers this question as he shares his thoughts on the subject while reflecting on the power of open-source principles and systems engineering as the keys to success in CrossTalk's lively and entertaining "Interview with Grady Booch." Grady also discusses the evolution of UML, the constraints of the DoD software world, the stagnation of languages, and the benefits of interdisciplinary training of our teams.

CrossTalk also offers a healthy line-up of informative articles exploring architectural topics such as how risk-driven models can guide architecture developers by mitigating risks using design techniques in the article "Just Enough Architecture: The Risk-Driven Model," and how acquisition and architecture can all altruistically co-exist in the article "Service Incentive: Towards an SOA Friendly Acquisition Process." See the practices used by the U.S. Army to promote architectural practices while increasing acquisition workforce talent by establishing Chief Software Architects in the article "The Chief Software Architect in U.S. Army Acquisition" and the intelligence community's efforts to advance architectural practices while satisfying tough requirements in "XDDS: A Scalable Guard-agnostic Cross Domain Discovery Service." Two final articles titled "Enabling Agility through Architecture" and "Agile Integration of Complex Systems" explore the practice of maintaining flexibility and innovation while implementing sound architectures.

"Global Workforce Development Projects in Software Engineering" is our final article which further explores the software engineering education concerns identified earlier in this issue by Grady Booch. The authors share information on two current software community projects: *Integrated Software and Systems Engineering Curriculum* and *The Body of Knowledge and Curriculum to Advance Systems Engineering*, both of which are aimed at garnering an agreement on how to educate, guide, and certify the systems and software engineering workforce. The community includes DoD, INCOSE, IEEE Systems Council, IEEE Computer Society Educational Activities Board, and the Association for Computing Machinery. Anyone related to this community should not miss this update.

The authors of this issue have provided me additional fodder to go on waxing poetic in regard to our theme. Again, call me a geek, but there is certainly something to be said for taking the time to contemplatively reach out to stakeholders and design a masterful architecture today.



Interview with Grady Booch

CrossTalk: Who are your most significant influences both inside and outside of software?

Well, I've got quite a few, so I'm sort of going to give this to you in a historical perspective.

U.S. Navy Rear Admiral Grace Murray Hopper, a fascinating woman I met some years ago, I still have my nanoseconds from her[1]—and readers who know about that will smile gently. In her lectures, she had this wonderful visual cue she used when telling people about the amazing things in regards to the shrinking of machines. She would have, prior to the lecture, taken telephone wire and cut it into little 11-inch segments and passed it out to people, saying, "Here's a nanosecond." It actually represented the distance that light would travel within a nanosecond—an amazing visualization. I was always touched by her grace, her ability to speak power to truth, and her ability to integrate the technical with the social.

Fred Brooks, of course, has been a tremendous influence to me. Ed Yourdon and Tom DeMarco on the process side, and more recently, folks like Kent Beck, Ward Cunningham and Scott Ambler. I also add to this list Mary Shaw at Carnegie Mellon—she has taught me a great deal about architecture, Philippe Kruchten did as well. The late Randy Pausch;[2] I had the delightful opportunity to meet with him briefly at a conference while he was presenting Alice.[3] And George Walther, who was my instructor at the Air Force Academy. He was the first person who really introduced me to the notion of discovering beauty in software.

Of course Jim and Ivar were tremendous influences on me—we could not have produced the UML without the collaboration from all three of us. We are three radically different personalities and, as I have said publicly, it's amazing that we accomplished what we did without felonies being committed along the way. But I'm delighted that we did, and I honor and respect them and dearly love the time I spent with them. It was a high point in my career.

Outside the software world, Richard Feynman is my absolute hero. His ability to just be a renaissance man, his interest in so many wide-ranging fields, his desire to follow his bliss … he is a role model for me.

CrossTalk: You have written numerous books in your career. Which one do you believe has had the most influence on the DoD software community?

Which one do I think has the most effect? The next one I'm writing. [LAUGHTER]

The next one is titled *The Handbook of Software Architecture*.[4] Again, it goes in my theme of architecture as an artifact and the important role I believe in delivering complex systems. My goal here is to basically document the architecture of 100 interesting systems and describe them. My intent is to capture what we find to be the best practices in architectural patterns that are out there. This is an effort that has been going on for seven years—and I hope I will finish it within the next seven years. It's hard research, but I'm learning a lot of things, discovering things, and inventing things along the way.

As for past work, probably *Object-Oriented Analysis and Design with Applications*[5] was the most significant one because it sort of helped start the effort of unification of the work Jim, Ivar, and I did. It was influential in terms of notation as well as process, and, frankly, in making object-oriented design a household name.

CrossTalk: What is the current impression of the future of the UML—a language you helped create?

Well this is a very timely question because we [at IBM] recently submitted to Object Management Group (OMG) a response to their request for information about the next generation of the UML.[6] I'll begin by saying where I think the UML is, and where the trajectory is going.

First, as one of the original officers of the UML, I am flattered, amazed, stunned, and staggered at the reach the UML has had. It has shown up in places I never, ever anticipated when Jim [Rumbaugh], Ivar [Jacobson], and I began the journey unifying our methods.

What delights me and absolutely tickles me is the realization that the goals of the UML are still very valid today, and UML 2.0 continues to help deliver in that regard. I see the UML being used in places far beyond whatever I anticipated and that is very exciting and very humbling.

Yet that being said, part of our recommendation back to the OMG and the biggest thing I pushed is for a return to the fundamental roots of the UML, which is really two-fold.

First, UML 2.0 is more complex than it needs to be, and I would like to see the UML become simplified over time. And that's not a means of throwing things out and not being backward compatible, it's just a matter of refactoring the language so that there is a common underlining core.

The other thing I would really like to see is return to the roots of the language not being a visual programming language, which has fueled a lot of the model-driven development work. In some domains, it's quite appropriate … but it's a modeling

language … I would like to increase the use in the semantics of the UML relative to things like reverse engineering and mining and reasoning about things as they unfold over time.

One other thing—in terms of where the UML is headed—is that I was blown away recently when I discovered an article called "The Systems Biology Graphical Notation."[7] Apparently it was inspired by the UML as an attempt to build a standard for biologists for modeling things within their world—things like mechanisms within cells and the like. So that's an example of where the UML has extended its reach far beyond whatever I imagined. That's pretty cool, and it also tells me that the language does have staying power; it's going to be around here for a long, long time. We do need to simplify and refactor it.

CrossTalk: If you were in charge of DoD's weapon systems software and infrastructure IT systems, what would be your top initiatives?

It really used to be, decades ago, that the DoD was leading the marketplace in the delivery of software-intensive systems. The harsh reality is that the commercial sector is leading best practices and really pushing the arc relative to software engineering and software development. So, in that regard, the DoD is behind the times. That is not to say that they are not pushing the limits in some areas. The kind of complexity we see in certain weapons systems far exceeds anything one would see commercially, but ultimately, there are a lot of things that the DoD can learn from the commercial world. As I look across the spectrum of systems that are successful and try to find the anti-patterns from those that are unsuccessful, there are three that come to mind and appear to have relevance for success—not necessarily in any order.

There's the leveraging of open-source principles. I know that the DoD has Forge.mil, which is evolving those many ideas of SourceForge, and I very much encourage that notion because there's this opportunity for transparency, visibility of software intensive systems—it has certainly added value in the commercial space. So I would certainly encourage and intensify the use of those open-source platforms.

The next initiative I would bring about would be the collaboration infrastructures. The reality is that the DoD builds software-intensive systems with contractors who are spread across the globe, potentially—and certainly the deployment of these systems is across the globe as well. I'm not sure that the DoD has invested enough. And it's not just the classic Web 2.0 kinds of things like wikis and shared whiteboards and the like. I would also do some exploration in virtual worlds, the kinds of things IBM and myself are trying to push in that space.

The third thing—and I've had some strong initiatives in this—is the whole area of architecture. What drives me to this conclusion is that as I look at the main complaints and pains that virtually every organization has in delivering software-intensive systems, there appears to be a common thread between the architecture and the artifact. So I would go beyond DoDAF [Department of Defense Architecture Framework]. I really like the standard. I think it's effective for what it's intended to be for—really trying to model the enterprise of the warfighter—but, in my personal opinion, I am less confident that it's appropriate for the architecture of the software-intensive systems. So I would certainly begin some initiatives to push for the notion of architecture as an artifact in terms of its representation and its governance of the social organizations around it.

CrossTalk: What is the next big approach to creating software-based systems that is going to make a significant difference?

In terms of the next big approach, I believe it is growth in our understanding of systems engineering.

Traditionally you begin the design saying, "I've got these pieces and let's throw in a processor here and there, and then you software guys go off and do your thing." The problem is you can't, from a systems engineering perspective, treat software as something you can put aside. Rather, it is an intrinsic, essential, universal piece of the system. So I think the biggest change we will see—or the biggest need—is the move toward a recognition that systems engineering needs to incorporate more and more of the practices we know into pure software systems because, in the warfighter's case, these are hardware/software systems—and that means we have to approach them differently than we have in the past.

So how does that manifest itself in terms of actionable things? The real news is that there is work to be done. INCOSE's beginning to embrace these ideas in the emergences of languages like SysML [the Systems Modeling Language] is helping us move along in that direction. But we don't know all the answers, and we're on a journey along the way—that's why I say it's the next big thing we'll have to worry about.

CrossTalk: What new advances and changes in languages and software engineering are on the horizon?

The following is, again, my personal opinion—not that of anybody living or dead or yet to be born, and I say this because it is a controversial one. I've said it publicly and usually I get lots of nasty e-mails after I say it, but my observation is that on the language side we're really at a plateau.

While I tracked what was happening in the language research space, I was really excited about what was going on in aspect-oriented programming. But that seems to have died out, in the sense that people were still dealing with problems in the weeds and it really hadn't risen up to the level of aspects at a higher level of abstraction. So on the language side, I think we're going to see a continuation in most of our existing languages. Look at C++ and you'll see that they have fixed a number of things in the current standards and they've really tried to extend it in some other areas as well—and these are largely incremental, albeit, important changes.

Where I think the biggest changes will happen will be back in the software engineering side. But before I attend to that, let's talk about what pushes us in that direction: What are the forces that cause that change?

There's the presence of legacy and how one addresses that. We have a crushing burden of legacy upon us—and not to put this in a negative light—but the reality is there is a significant capital investment in legacy and that leads us to not

throw these things away, but rather trying to figure out how to interoperate with them. SOA [service-oriented architecture] certainly plays an important role helping them interoperate, but then again legacy—the forces around that—are one issue. Another thing that is pushing up is presence of multi-core. The frequency-scaling wars are over, so we can only begin to boost computational resources by boosting frequencies certain ways. If you have an obviously decomposable parallel problem, multi-core usually fits, but you have a less than obvious decomposition. It's really nasty and hard. Another force I think we're driving up is the whole problem with security—and then the biggest one, perhaps the most dominant to impact us, is the issue of complexity. We are building systems of crushing complexity, so we need some help in that regard.

Those things together I think are pushing us. With regards to what is happening in the software engineering side, the good news is I think we have a good picture for how high-ceremony processes and agile processes work well together. So there's a lot of good information coming out of that world. And, although I think it may be self-serving, I see short-term growth towards the practices around architecture as an artifact, and then the next thing on the horizon is less-so software engineering and more-so systems engineering.

CrossTalk: What are the restraining parameters that hold software engineering back from more breakthroughs?

First off, I don't really believe in breakthroughs. The reality of the progress of science, especially in software, is that changes come from the confluence of many things, where you might reach a tipping point that changes things. But I'm more of one for evolution than revolution. Frankly, I even consider object-oriented design to be an evolutionary thing as opposed to a revolutionary thing.

I think I've actually talked about the true restraining factors already: legacy, inoperability, multi-core security. And the last is complexity. We are dealing with systems that far exceed the intellectual capacity of any single human. We generally lack the notations, processes and measurements to help us deal with that complexity—so that's what holds us back. It's a wickedly hard problem.

CrossTalk: What are we to do with all the DoD systems that were implemented in older object-oriented languages like Ada, Modula, etc., as there are less and less engineers skilled to enhance and maintain the code?

There are older object-oriented languages being used. In fact, I'm engaged in a project that is still using Ada, and I'm excited that they are because it is really well-proven language.

It is a problem, but not one the DoD has alone. I had been working on a project with the IRS … a system that is central to their tax processing has about 500,000 lines of assembly language, a lot of which was written in the '60s and it still exists there. I see systems written in COBOL; I've seen systems written in PL/I. So this is a systemic problem that goes to the heart of the issue of legacy of older systems that I mentioned.

I've actually written and discussed this very topic; what I call the "Nine Things You Can Do with Old Software."[8] One thing you can do is harvesting—which is basically taking these older things and doing the reverse engineering of pieces of them to extract the algorithms, the data structures, things like that, and then rewriting them—but that is so very hard. Another—the most effective thing that I have seen—is the notion of continuous architectural transformation. It requires considerable process discipline and it goes back to the heart of architecture and artifacts. Only a few organizations that I've seen have been really successful. I hold up eBay as a classic example.

CrossTalk: That's interesting. I didn't know there was that much old language out there still being utilized.

Oh yeah, there are gobs of languages. I did a quick calculation asking how many lines of codes do we produce in the world on a yearly basis? It was a low number, but if you make an estimate for the number of software professionals, the number of people that actually code, the number of lines of code per average per year, you end up with around 33 billion lines of code new or modified or produced every year—and I will be honest in saying that's conservative and it's probably off by an order of magnitude. So if you integrate that over the years, it means, at the very least, that we probably have over a trillion lines of code out there—and much of that is still running in these old systems. So the presence of these legacy systems is a reality—and it's not just a problem the DoD has.

CrossTalk: What major changes would you like to see in the DoD to forward software engineering success?

I think the major change is in education. I don't mean to be critical, but in many ways the DoD's expertise has, frankly, been outsourced to its contractors. It is not to say that is a horrible, terrible thing, but a lot of the things that happened in old warfighting systems came through intrinsic expertise inside the DoD. I would strongly encourage the increase of education of the DoD's intrinsic forces with regards to decision engineering and software engineering—and draw back into the DoD more of that intellectual property. Ultimately, delivering for the warfighters is what the DoD is all about, and that requires an intensely educated staff to make that happen. How does one make that manifest? I think there is work to be done in acquisition policy, in processes for delivery in the use of things like DoDAF. I think the DoD itself can lead and should lead this, and it needs to make this change in the interspatial spaces of its training, in its service academies, and in its colleges as well.

CrossTalk: Have you seen anything to suggest that the DoD has gathered that same point of view and that it might be starting to change its perspective and train people differently?

Walt J. Okon and I recently had a conversation on that very topic. I did raise with him the notion of education. I was absolutely ready to dance on the table when he told me that one of his major initiatives, beyond 2.0 reaching closure, is that whole issue of education. Beyond what he is doing, I don't have a lot of insight, but I am certainly encouraged by his efforts.

CrossTalk: Back to the idea of needed training prior to getting ensconced in the industry: How do you see the current state of software engineering in higher education, and where do you think it needs to go?

I've had the delightful opportunity to engage with a lot of different schools. I make a yearly jaunt around the universities—both in the U.S. and other places in the world—to give lectures and the like. I've also had the chance to interact with people both in the ACM [Association for Computing Machinery] and in the IEEE on K-12 and undergraduate and graduate degrees.

What is growing are the interdisciplinary kinds of things like I've seen at CMU, and at USC through Barry Boehm, where systems engineering is coming together and software is an important piece of that.

There is this mental model I use that I speak of as "the laws of software." So if you imagine that we have a surplus of cognitive resources—in other words, human intelligence or human imagination is not a limited resource—we come up with these visions and we have to turn that into "raw running naked" code. The question for me is what separates us from vision, to turn that into raw running naked code—and the answer is there are these things in the laws of software.

You'll see that things move from the computer science-y things, which are very mathematically based and very fundamental, into the things that become more human-oriented—elements like politics and ethics and moral issues. We think we know how to build certain things … the question is should we?

What makes it most difficult to move from vision to execution is something that swirls around the problems of design and the problems of organization. How do I best architect a system? How do I best architect my organization to deliver that system? As it turns out, there's this wonderful, delicious cusp of the technical and the social, and that's where the sweet spot for delivery is in education. How does one attend to the fiercely technical problems, but at the same time be cognitive of the social issues as well? I swear there are days that I go into an organization where I'll show up as über geek and other days I have to show up as Dr. Phil, slapping faces around, saying, "My God, what are you thinking?" So, in terms of where I think things need to go—well, for people delivering software-intensive systems, I think our education system has to attend to that dance between the technical and the social.

So this is still an exciting place to be. I encourage people who are thinking about this field to recognize that there are a lot of wickedly entertaining, exciting and delicious problems to solve. We're not done yet.

CrossTalk: I was recently in a conversation where we were trying to set up a degree program with a local university for UAS [unmanned aerial systems] and the big argument was hardcore engineers versus interdisciplinary people. I take it you're leaning toward interdisciplinary as a strength?

Well, I say it is very much a strength because if you look at unmanned vehicles, this is a classic systems-engineering problem. There are some wickedly technical problems to overcome, but ultimately I'm delivering a system to be used by humans, to be used in the context of other complex warfighting systems. These are not islands, so I would want to seek out the best ideas from a variety of places. So yes, I can't imagine one considering this other than interdisciplinary activity.

Through the mixtures of putting smart people together in different domains, innovation comes about in unexpected ways.

The final thing I'd offer is, you know, that this is still an exciting discipline. The global economy is in a funk, there's no doubt about it. I've been lecturing recently about the notion of software abundance in the space of economic scarcity, and I'm utterly convinced that the delivery of software-intensive systems is still a major source of innovation and, therefore, economic growth. So this is still an exciting place to be. I encourage people who are thinking about this field to recognize that there are a lot of wickedly entertaining, exciting and delicious problems to solve. We're not done yet.

NOTES

1. To learn more about Rear Admiral Hopper (1906-1992)–including her famed nanoseconds–visit <www.chips.navy.mil/links/grace_hopper/womn.htm>.
2. Pausch may be known best for his Last Lecture: "Really Achieving Your Childhood Dreams."
3. Alice is a 3-D programming environment.
4. Currently, Booch maintains a blog, <www.handbookofsoftwarearchitecture.com>, for The Handbook of Software Architecture, which serves as the repository for ongoing work in an effort that will eventually be published in print.
5. First published in 1991, Booch's book is in its third edition (2007).
6. See < www.uml.org> to learn more about UML, the current status of UML 2.0, and the role of the OMG.
7. By Nicolas Le Novère, et al., in the 7 Aug. 2009 edition of *Nature Biotechnology*. The article is available at <www.nature.com/nbt/journal/v27/n8/full/nbt.1558.html#a1>.
8. See the Sept./Oct. 2008 edition of *IEEE Software* or listen to the podcast, "Nine Things You Can Do With Old Software," at <www.computer.org/portal/web/computingnow/onarchitecture>.

Just Enough Architecture: The Risk-Driven Model

George Fairbanks
Rhino Research

Abstract: Developers have access to more architectural design techniques than they can afford to apply. The Risk-Driven Model guides developers to do just enough architecture by identifying their project's most pressing risks and applying only architecture and design techniques that mitigate them. The key element of the Risk-Driven Model is the promotion of risk to prominence. It is possible to apply the Risk-Driven Model to essentially any software development process, such as waterfall or agile, while still keeping within its spirit.

If you knew nothing about software development, you might imagine that the best developers were the ones who spent the most time writing code. Yet it has been clear for a long time that judicious application of other activities—such as analysis, design and testing—will result in better software. However, at some point, doing more analysis, design or testing becomes counterproductive since it steals time and resources away from other, more productive activities.

Designing an appropriate software architecture is one of those non-coding activities that can improve the quality of a system, but if developers spend too much effort on it, they will be stealing from other activities. Consequently, an important question is raised: How much design and architecture should developers do? Any realistic answer must balance design and architecture effort against other activities.

This article introduces the Risk-Driven Model of architectural design. It guides developers to apply effort to their software architecture commensurate with the risks faced by their project. That is, low-risk and highly precedented systems should skimp on architecture, while high-risk and novel systems should pay more attention to it.

This might seem like common sense, but it is not what happens today on most projects. Most commonly, a project's software development process dictates both the amount of effort and the specific architecture techniques. Unless this process is tuned to risk, it results in too much or too little effort spent on architecture.

How Architecture is Done Today

There is active debate about how much architecture work developers should do and several answers have been proposed:

>> No architectural design. Developers should just write code. Design happens, but is coincident with coding, and happens at the keyboard rather than in advance.

>> Use a yardstick. For example, developers should spend 10% of their time on architecture and design, 40% coding, 20% integrating and 30% testing.

>> Build a documentation package. Developers should employ a comprehensive set of design and documentation techniques sufficient to produce a complete written design document.

Any of these answers could be appropriate, but it depends on the project. The problem with these answers is that they do not help developers find a balance—they instead prescribe that balance in advance. What developers need is a way to decide which architecture techniques they should apply and which they should skip.

The Risk-Driven Model

The Risk-Driven Model helps developers decide how much architecture work to do. The essence of the Risk-Driven Model is these three steps:

1) Identify and prioritize risks
2) Select and apply a set of architecture techniques
3) Evaluate risk reduction

It helps developers follow a middle path, one that avoids wasting time on techniques that help their projects only a little, but ensures that project-threatening risks are addressed by appropriate techniques.

The key element of the Risk-Driven Model is the promotion of risk to prominence. What you choose to promote has an impact. Most developers already think about risks, but they think about lots of other things too, and consequently risks can be overlooked.

Projects face different risks, so they need different architecture techniques. Some projects will have tricky quality attribute requirements that need up-front planned design, while other projects need tweaks to existing systems and entail little risk of failure. Some development teams are distributed, so they document their designs for others to read, while other teams are co-located and can write fewer documents.

Technique Choices Should Vary

Most organizations guide developers to follow a process with some kind of documentation template or a list of design activities. Templates can be beneficial and effective, but they can also inadvertently steer developers astray. Here are some examples of well-intentioned rules that guide developers to activities that may be mismatched with their project's risks:

>> The team must always (or never) build a full documentation package for each system.

>> The team must always (or never) build a class diagram, a layer diagram, etc.

>> The team must spend 10% (or 0%) of the project time on design or architecture.

It would be a great coincidence if an unchanging set of diagrams or techniques was always the best way to mitigate a changing set of risks. Standard processes or templates can be helpful, but they are often used poorly. Over time, you may be able to generalize the risks on the projects at your organization and devise a list of appropriate techniques. The important part is that the techniques match the risks.

Example Mismatch

Imagine an organization that builds a three-tier system. The first tier has the user interface, and is exposed to the internet. The biggest risks might be usability and security. The second and third tiers implement business rules and persistence; they are behind a firewall. The biggest risks might be throughput and scalability.

What often happens is that both teams follow the same company-standard process or template and produce, say, a module dependency diagram. The diagram is probably somewhat helpful, but it takes the space of more helpful techniques better matched to the project risks.

Following the Risk-Driven Model, the front-end and back-end teams would apply different techniques. For example, the front-end developers might create user interface mockups and analyze their design for intrusion vectors. The back-end developers might do performance modeling and impose constraints to enable scalability.

Are You Risk-Driven Now?

Many developers believe that they already follow a Risk-Driven Model, or something close to it. Yet there are telltale signs that many are not.

One sign is an inability to list the risks they confront and the corresponding techniques they are applying. Any developer can answer the question, "Which features are you working on?" but many have trouble with the question, "What are your primary failure risks and corresponding engineering techniques?" If risks were indeed primary, it would be an easy question to answer. Another sign is that all developers use the same techniques.

Most architecture templates have a section on risks, but that is not the same as using risks to decide which techniques to use. To be risk-driven in your architectural decision making, you need to have a rationale that ties your actions (i.e., use of architectural techniques) back to your risks.

Logical Rationale

The Risk-Driven Model has the useful property of yielding arguments that can be evaluated. An example argument would take this form:

We identified A, B and C as risks, with B being primary. We spent time applying techniques X and Y because we believed they would help us reduce the risk of B. We evaluated the resulting design and decided that we had sufficiently mitigated the risk of B, so we proceeded on to coding.

Other developers might disagree with this assessment, so they could provide a differing argument with the same form, perhaps suggesting that risk D be included. A productive, engineering-based discussion of the risks and techniques can ensue because the rationale behind your opinion has been articulated and can be evaluated.

Incomplete Architecture Designs

When developers apply the Risk-Driven Model, they only design the areas where they perceive failure risks. Most of the time, applying a design technique means building a model of some kind, either on paper or a whiteboard. Consequently, the architecture model will likely be detailed in some areas and sketchy, or even non-existent, in others.

For example, if developers have identified some performance risks and no security risks, they would build models to address the performance risks but those models would have no security details in them. Still, not every detail about performance would be modeled and decided. Remember that models are an intermediate product and developers can stop working on them once they have become convinced that the architecture is suitable for addressing the risks.

Software Processes

Since the Risk-Driven Model applies only to architecture design and is not a full software development process, it is possible to apply it within essentially any process. A waterfall process prescribes planned design in its analysis and design phases, but does not tell you what kind of architecture and design work to do, or how much to do. You can apply the Risk-Driven Model during the analysis and design phases to answer those questions.

Today, many developers follow iterative processes, and it goes against the grain to bolt-on a several-month architecture design phase. An iterative process does not have a designated place for design work, but architecture design could be done at the beginning of each iteration. The amount of time spent on design would vary based on the risks. Figure 1 provides a notional example of how the amount of design could vary across iterations based on your perception of the risks.

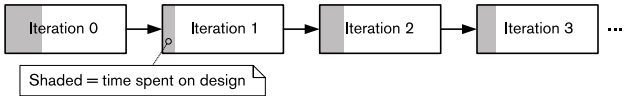


Figure 1: An example of how the amount of design could vary across iterations based on your perception of the risks. In this example, more risk was perceived in iterations 0 and 2.

ADDITIONAL READING

Barry Boehm wrote about risk in the context of software development with his paper on the Spiral Model of software development [1]. The Risk-Driven Model would, on first glance, appear to be quite similar to the Spiral Model of software development, but the Spiral Model applies to the entire development process, not just the design activity.

The Unified Process and its specialization, the Rational Unified Process, are iterative, spiral processes [2, 3]. They highlight both the importance of addressing risks early and the use of architecture to address risks. The (R)UP advocates working on architecturally relevant requirements first, in early iterations.

Barry Boehm and Richard Turner discuss risk and agile processes [4] and the summary of their judgment is, "The essence of using risk to balance agility and discipline is to apply one simple question to nearly every facet of process within a project: Is it riskier for me to apply (more of) this process component or to refrain from applying it?"

The Risk-Driven Model is similar to global analysis as described by Christine Hofmeister, Robert Nord and Dilip Soni [5]. The intention of global analysis is not to optimize the amount of effort spent on architecture, but rather to ensure that all factors have been investigated.

This article is excerpted from a chapter in the book Just Enough Software Architecture: A Risk-Driven Approach and the full chapter is available for download [6]. It additionally discusses engineering versus management risks, and details on application of the Risk-Driven Model to various software development processes.

Agile processes are usually special cases of iterative processes, but they have the additional difficulty of fitting architectural work into a backlog. If the backlog must now contain both user features and technical risks, it may be difficult for business stakeholders to prioritize it.

A Spiral process and the Risk-Driven Model are cousins in that risk is primary in both. The difference is that the Spiral process, being a full software development process, prioritizes both management and engineering risks and guides what happens across iterations. The Risk-Driven Model only guides design work to mitigate engineering risks, meaning that it would help you understand which architecture techniques you should use within a specific iteration of the Spiral process. Applying the Risk-Driven Model to a Spiral process or the (Rational) Unified Process works the same as with an iterative process.

Guidance On Choosing Techniques

So far, you have been introduced to the Risk-Driven Model and have been advised to choose techniques based on your risks. You should be wondering how to make good choices. In the future, perhaps a developer choosing techniques will act much like a mechanical engineer who chooses materials by referencing tables of properties and making quantitative decisions. For now, such tables do not exist.

However, there are principles that underlie any table or any veteran's experience, principles that explain why technique X works to mitigate risk Y. Here is a brief preview:

First, sometimes you have a problem to find while other times you have a problem to prove, and your technique choice should match that need. Second, some problems can be solved with an analogic model while others require an analytic model, so you will need to differentiate these kinds of models. And third, some techniques have affinities, like pounding is suitable for nails and twisting is suitable for screws.

Problems to Find and Prove

In his book How to Solve It, George Polya identifies two distinct kinds of math problems: problems to find and problems to prove [7]. The problem, "Is there a number that when squared equals 4?" is a problem to find, and you can test your proposed answer easily. On the other hand, "Is the set of prime numbers infinite?" is a problem to prove. Finding things tends to be easier than proving things because for proof, you need to demonstrate something is true in all possible cases.

When searching for a technique to address a risk, you can often eliminate many possible techniques because they answer the wrong kind of Polya question. Some risks are specific, so they can be tested with straightforward test cases. It is easy to imagine writing a test case for "Can the database hold names up to 100 characters?" since it is a problem to find. Similarly, you may need to design a scalable website. This is also a problem to find because you only need to design (i.e., find) one solution, not demonstrate that your design is optimal.

Conversely, it is hard to imagine a small set of test cases providing persuasive evidence when you have a problem to prove. Consider, "Does the system always conform to the framework Application Programming Interface?" Your tests could succeed, but there could be a case you have not yet seen, perhaps when a framework call unexpectedly passes a null reference. Another example of a problem to prove is deadlock: Any number of tests can run successfully without revealing a problem in a locking protocol.

Analytic and Analogic Models

Michael Jackson, crediting Russell Ackoff, distinguishes between analogic models and analytic models [8]. In an analogic model, each model element has an analogue in the domain of interest. A radar screen is an analogic model of some terrain, where blips on the screen correspond to airplanes—the blip and the airplane are analogues.

Analogic models support analysis only indirectly, and usually domain knowledge or human reasoning are required. A radar screen can help you answer the question, "Are these planes on a collision course?" but to do so you are using your special purpose brainpower in the same way that an outfielder can tell if he is in position to catch a fly ball.

An analytic model, by contrast, directly supports computational analysis. Mathematical equations are examples of analytic models, as are state machines. You could imagine an analytic model of the airplanes where each is represented by a vector. Mathematics provides an analytic capability to relate the vectors, so you could quantitatively answer questions about collision courses.

When you model software, you invariably use symbols, whether they are Unified Modeling Language (UML) elements or some other notation. You must be careful because some of those symbolic models support analytic reasoning while others support analogic reasoning, even when they use the same notation. For example, two different UML models could represent airplanes as classes, one with and one without an attribute for the airplane's vector. The UML model with the vector enables you to compute a collision course, so it is an analytic model. The UML model without the vector does not, so it is an analogic model. So simply using a defined notation, like UML, does not guarantee that your models will be analytic. Architecture Description Languages are more constrained than UML, with the intention of nudging your architecture models to be analytic ones.

When you know what risks you want to mitigate, you can appropriately choose an analytic or analogic model. For example, if you are concerned that your engineers may not understand the relationships between domain entities, you may build an analogic model in UML and confirm it with domain experts. Conversely, if you need to calculate response time distributions, then you will want an analytic model.

Techniques With Affinities

In the physical world, tools are designed for a purpose: hammers are for pounding nails, screwdrivers are for turning screws and saws are for cutting. You may sometimes hammer a screw, or use a screwdriver as a pry bar, but the results are better when you use the tool that matches the job.

In software architecture, some techniques only go with particular risks because they were designed that way and it is difficult to use them for another purpose. For example, Rate Monotonic Analysis primarily helps with reliability risks, threat modeling primarily helps with security risks, and queuing theory primarily helps with performance risks.

Conclusion

This article introduces the Risk-Driven Model that encourages developers to: (1) prioritize the risks they face, (2) choose appropriate architecture techniques to mitigate those risks, and (3) re-evaluate remaining risks. It encourages just enough software architecture by guiding developers to a prioritized subset of architecture activities. Design can happen up-front but it also happens during a project. Low-risk projects can succeed without any planned architecture work, while many high-risk projects would fail without it.

The Risk-Driven Model walks a middle path that avoids the extremes of complete architecture documentation packages and complete architecture avoidance. It follows the principle that your architecture efforts should be commensurate with the risk of failure. The key element of the Risk-Driven Model is the promotion of risk to prominence. Each project will have a different set of risks, so each will need a different set of techniques. To avoid wasting your time and money, you should choose architecture techniques that best reduce your prioritized list of risks.

ABOUT THE AUTHOR

George Fairbanks is the president of Rhino Research, a software architecture training and consulting company. His new book, Just Enough Software Architecture: A Risk-Driven Approach, has been widely praised by academics and practicing software developers. His Ph.D. work at Carnegie Mellon University introduced design fragments, a new way to specify and assure the correct use of frameworks through static analysis. He has been teaching software architecture and object-oriented design for over a decade. He has written code for telephone switches, the Eclipse IDE, Android apps and his own web dot-com startup.

Rhino Research
george.fairbanks@rhinoresearch.com
2445 7th St, Boulder CO 80304
(303) 834-7760

REFERENCES

1. Boehm, Barry. "A Spiral Model of Software Development and Enhancement." 21.5 *IEEE Computer* (1988): 61–72.
2. Jacobson, Ivar, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Professional, 1999.
3. Kruchten, Philippe. *The Rational Unified Process: An Introduction*. Addison-Wesley Professional, 2003.
4. Boehm, Barry and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Professional, 2003.
5. Hofmeister, Christine, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Addison-Wesley, 2000.
6. Fairbanks, George. *Just Enough Software Architecture: A Risk-Driven Approach* <http://RhinoResearch.com/book>. Marshall & Brainerd, 2010.
7. Polya, George. *How To Solve It: A New Aspect of Mathematical Method*. Princeton University Press. 2004.
8. Jackson, Michael. *Software Requirements and Specifications*. Addison-Wesley. 1995.

Enabling Agility Through Architecture

Nanette Brown, Robert Nord, Ipek Ozkaya
Software Engineering Institute, Carnegie Mellon University

Abstract: Industry and government stakeholders continue to demand increasingly rapid innovation and the ability to adjust products and systems to emerging needs. Amongst all the enthusiasm for using Agile practices to meet these needs, the critical role of the underlying architecture is often overlooked.

Time frames for new feature releases continue to shorten, as exemplified by Z. Lemnios, Director of Defense Research and Engineering:

"Get me an 80% solution NOW rather than a 100% solution two years from now and help me innovate in the field" [1].

To meet these demands, government and government contractors are now looking closely into the adoption of agile practices [2] [3].

End users demand Enhancement Agility, the ability to keep adjusting the product to emerging needs through the addition of new features. Existing approaches to achieving Enhancement Agility vary, depending upon the lifecycle under which the product or system is being developed.

Under the Waterfall paradigm of software development, an extensive requirements phase is conducted to anticipate needs for both the initial and subsequent releases of the product or system being developed. Following the requirements phase, an architecture phase is conducted to develop a comprehensive underlying technical infrastructure. Within the Waterfall model, once the architecture is implemented, Enhancement Agility can be achieved, provided that the emergent user needs fit within the boundaries anticipated during the requirements phase.

However, taking the Waterfall approach presents two potential problems. First, when working in a new, unknown emergent problem space, building an architectural platform that reliably anticipates all future needs is an extremely difficult undertaking. Secondly, under the Waterfall paradigm, considerable effort and expense is incurred before any actual value is achieved (i.e., before any features are delivered to the user).

In contrast to Waterfall methodologies, Agile software development methods focus on delivering observable benefits to the end users through working software, early and often. A backlog of functional "user stories" is created. These stories are prioritized by end users and/or the product owner, acting as the user advocate. Development teams draw stories from the backlog and implement them in accordance with an end-user prioritization scheme. The Agile community's focus on continuous delivery of user-valued stories is another means of achieving Enhancement Agility. However, this approach also has its shortfalls, stemming mainly from an inadequate focus on dependency analysis.

Individual stories cannot be regarded in isolation. Stories have dependencies on other stories. In Software by Numbers, Denne and Cleland-Huang use the term "greedy algorithm" to refer to a prioritization scheme which focuses strictly on implementing the story with the highest immediate value [4]. They point out that, at times, higher-value stories may depend upon (i.e., require prior implementation of) lower value stories. Thus, truly optimizing value to the user requires teams to look ahead and anticipate future needs.

Similarly, stories have dependencies upon the architectural elements of the system. These dependencies exist regardless of domain stability or technical maturity. They exist regardless of whether the system is in its initial development stages or has been deployed and has been in the field for several years. The ability to identify and analyze architectural dependencies and incorporate dependency awareness into a responsive development model exemplifies the notion of Architectural Agility. It is our thesis that without Architectural Agility, Enhancement Agility cannot be reliably sustained.

Architectural Agility and Release Planning

Architectural Agility addresses shortcomings that occur within both the Waterfall and the Agile lifecycle models. Architectural Agility allows architectural development to follow a "just-in-time" model. Delivery of customer-facing features is not delayed pending the completion of exhaustive requirements and design activities and reviews. At the same time, Architectural Agility maintains a steady and consistent focus on continuing architectural evolution in support of emerging customer-facing features. It avoids the pitfalls of a myopic focus on user stories, which over time can lead to increased complexity and "tortured" implementation choices as develop-

ers seek to incorporate features that the architecture was not designed to support. Proceeding under the latter paradigm leads to the all-too-familiar situation in which features gradually take longer and longer to implement, the code becomes more and more buggy, and eventually management is informed that the system must be scrapped and rewritten "from scratch."

Our mantra for Architectural Agility is "informed anticipation." The architecture should not over-anticipate emergent needs, delaying delivery of user value and risking development of overly complex and unneeded architectural constructs. At the same time, it should not under-anticipate future needs, risking feature development in the absence of architectural guidance and support. Architectural Agility requires "just enough" anticipation. To achieve the quality of being "just enough," architectural anticipation must be "informed." Dependency analysis, real options analysis and technical debt management are the tools through which "informed anticipation" can be achieved. The remainder of this article will illustrate the application of these techniques through the practice of release planning.

Figure 1 shows a release planning board that represents the typical heuristics used within the Agile community for release planning. Desired stakeholder capabilities are represented as "user stories." These user stories are allocated to iterations in order of their priority to the end user.

Figure 2 shows an enhanced release planning board that incorporates planning for development of the underlying software architecture. In addition to selecting stories to be developed within each iteration, the team identifies the architectural elements that must be implemented to support them. This version of the release planning board also incorporates a "technical research" activity, recognizing that architectural development frequently requires investigation and analysis of alternate approaches. Finally, the term "capabilities" has been used in place of "user stories," reflecting a need to consider non-functional, quality attribute requirements, as well as the need to incorporate requirements across a broad range of stakeholders.

As an example, consider the Apps for the Army initiative [5]. The ability to add new and innovative apps quickly and easily exemplifies the concept of Enhancement Agility. However, Architectural Agility is required to supply the underlying technical infrastructure to support the app-based development model. The app-based development model includes a developer framework and run-time infrastructure that are part of the notion of an app store.

A conceptual architecture for an app store is illustrated in Figure 3. This conceptual architecture describes the essential high-level architectural elements such as content management, service management, data access, security and a range of external target devices that can access/manipulate the apps. Using an agile approach of starting small and growing the system, the team selects capabilities that support a small number of predetermined apps in the early iterations. This requires identifying those architectural elements within the business logic, data access, and service management com-

Figure 1: Agile iteration planning – focus on User Stories

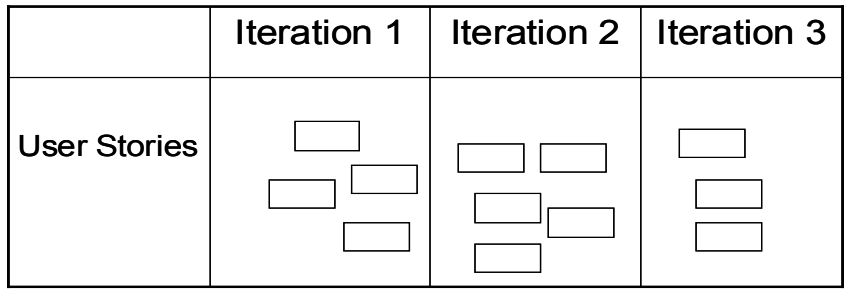


Figure 2: Architectural elements in agile iteration planning

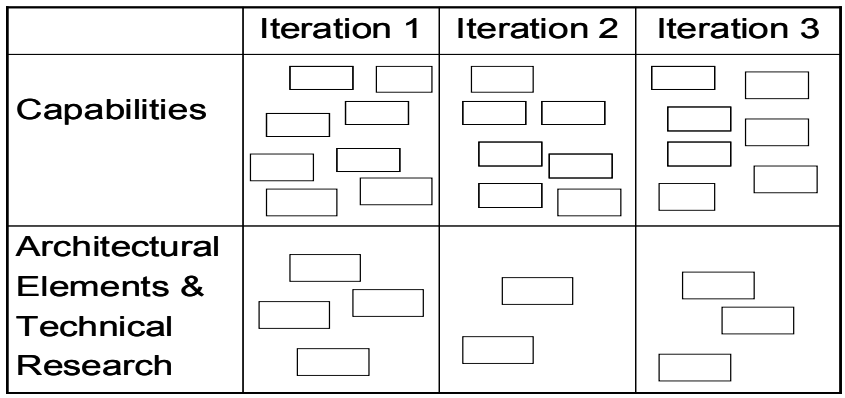
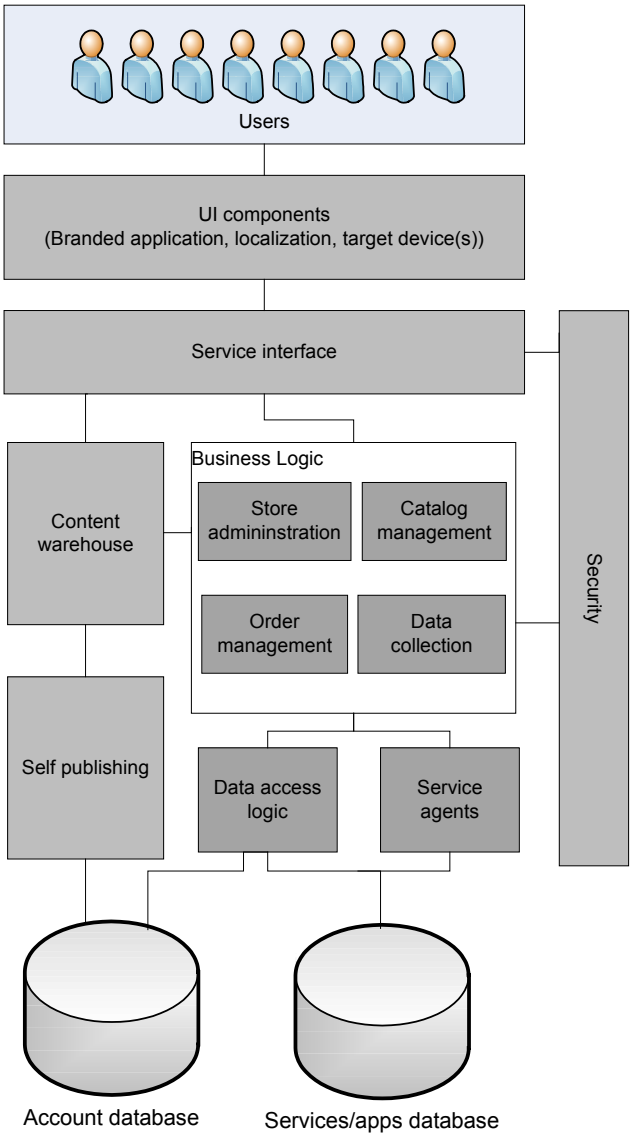
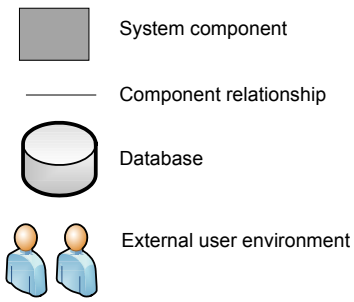


Figure 3: Conceptual App Store Architecture and High-Level Capability Dependencies



		Reporting	App catalog management	Sales management	Upgrade management	Promotion management	Order management	Account management	
Capabilities	Reporting		x	x	x	x	x	x	6
	App catalog management	x		x	x	x	x	x	6
	Sales management		x			x	x	x	4
	Upgrade management		x						1
	Promotion management	x	x	x			x	x	5
	Order management	x							1
	Account management						x		1
Architectural elements	UI components	x	x	x	x	x	x	x	7
	Service interface	x	x	x	x	x	x	x	7
	Content warehouse	x	x		x				3
	Self publishing	x	x		x				3
	Account database	x	x				x	x	4
	Services/apps database		x		x				2
	Data access logic		x				x	x	3
	Service agents		x		x	x	x	x	5
	Security		x	x	x	x	x	x	6
	Order management	x					x	x	3
	Data collection	x						x	2
	Store administration	x	x	x	x	x	x	x	7
	Catalog management	x	x		x				3
		9	11	4	9	5	8	9	



ponents that support these capabilities. In later iterations, the team expects to focus on scaling the system in the number of apps and users, enhancing security, and allowing users to contribute their own apps. Architectural elements within the security, content management, and publishing components need to be scrutinized to see which are needed to support these additional capabilities.

Implementing this type of planning heuristic requires the ability to do dynamic dependency management in a manner that is both rigorous and responsive. Dependencies between capabilities and architectural elements need to be identified for each iteration in order to prioritize and schedule work within a release.

Architecture Dependency Management

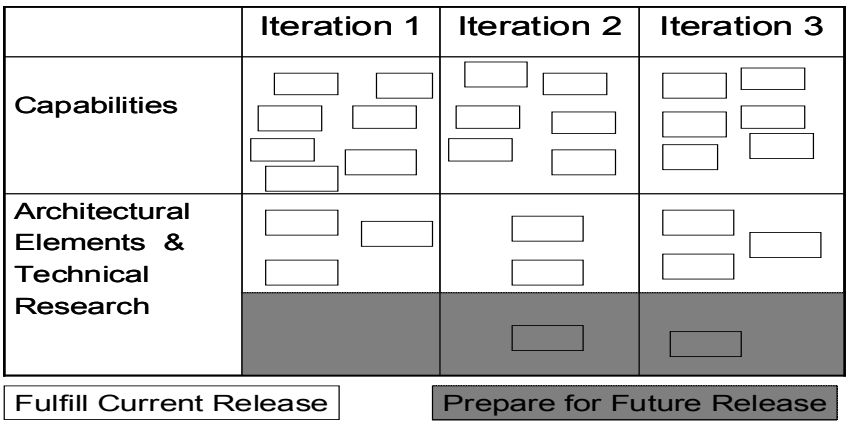
Dependency management has been studied extensively at the level of code artifacts. Applying dependency management at the architecture level is beginning to show promising results due to increasingly effective tool support. These metrics can be extracted from the architecture, represented in the form of a Dependency Structure Matrix (DSM). The DSM is a compact representation which lists all constituent subsystems/activities and the corresponding information exchange and dependency patterns. Domain Mapping Matrices (DMMs) augment DSM analyses and can be used to represent the dependencies between capabilities and architectural elements.

Returning to the example, dependency analysis for the app store must consider dependencies between capabilities

as well as dependencies between architectural elements and capabilities. These dependencies are identified in the matrix in Figure 3. The capabilities portion of this matrix is an example of a DSM. An X mark indicates that the capability in the row provides information to the capability in the column. Reading across the row labeled "App catalog management," it is clear that all other capabilities depend on it. The architectural elements portion of the matrix is an example of a DMM. A marked cell indicates that the architectural element in the row implements an aspect of the capability represented in the column. Reading down the column labeled "App catalog management," it becomes clear that the App catalog management capability depends on almost all of the architectural elements. Having this kind of view can be essential in focusing the iterations within releases.

Metrics associated with dependency also provide data for inferring the likely costs of change propagation, especially when dependencies between architectural elements are also considered (not shown in Figure 3). One such example is discussed in Carriere et al where the value of re-architecting decisions needed to be understood to determine if the expense to implement them was justified [6].

Figure 4: Informed anticipation in the context of agile release planning



Architecture Heuristics Focused on Value: Real Options Analysis and Technical Debt Management

For effective Architectural Agility, dependencies between capabilities and architectural elements need to be identified not only to fulfill the current release, but to plan for future releases as well (Figure 4). Informed anticipation requires incorporating architecture heuristics focused on value into the planning model. Real options analysis and technical debt management offer potential models to make an informed choice and find the right balance of agility, innovation, and speed on the one hand, and governance, flexibility, and planning for future needs on the other.

This additional set of considerations adds a new dimension to the release planning board. This added dimension allows the identification of architectural constructs that, while not required for the current release, should potentially be incorporated into the current release in anticipation of future stakeholder goals.

As an example, the initial number of deployed apps is expected to be small, so capabilities such as scalability could be deferred and assigned to a future release. However, it is also true that by setting up an app store scalability infrastructure—that is, buying the option of scaling up—you can reduce your technical debt down the road. By choosing to take a shortcut—not buying the option—you incur possible technical debt.

The question of how to optimally allocate architectural elements that deal with scalability to releases can benefit from applying real options analysis. Real options analysis is a financial analysis model to help determine whether some upfront cost should be spent (buying the option) to have the right, but not the obligation, to take an action in the future (exercising the option). The real options analysis method applies the financial options theory to quantify the value of flexibility in real assets and business decisions to determine the value of such delayed decision making. And both common sense and the theory demonstrate that the higher the uncertainty, the more it makes sense to wait to act and defer the decisions. From this perspective, the agile community has used the concept of real options in separating concerns that have immediacy and those that can possibly wait.

In agile release planning, real options analysis is a way to look at the allocation of architectural elements to releases based on their dependencies from the perspective of future value [7]. In architecture terms, taking an option could be applying an architecture pattern, providing a well-structured modular design that supports Enhancement Agility. Real options analysis can be informed and complemented by a consideration of technical debt.

The technical debt metaphor [8] highlights that doing things the quick and dirty way for short-term benefit sets us up with a technical debt. Like a financial debt, the technical debt incurs interest payments, which come in the form of the extra effort that we have to do in future development because of suboptimal design choices. We can choose to continue paying the interest, or we can pay down the principal by refactoring and improving the design. Although it costs to pay down the principal, we gain by reduced interest payments in the future.

Agile development methods aim to manage technical debt through refactoring practices. Refactoring is restructuring an existing body of code, altering its internal structure without changing its external behavior. However, when significant architectural change is needed, such small, local refactoring efforts cannot compensate for the lack of an architecture that is necessary to guide the architect in achieving the goals of the system. In this case, lack of Architecture Agility starts compromising Enhancement Agility.

Informed Anticipation Guiding Agile Release Planning

Unifying the concepts of technical debt, real options, and uncertainty management is a common focus on the question "Should I take a certain action today in anticipation of increased benefit and reduced cost in the future?" Taking the correct action today provides an option which can be acted upon in the future. This is where the agile mindset and architecture reasoning tend to diverge. Agile projects focus on stories that are needed in the current release and rely on code-level refactoring to incorporate future stories. However, relying only on code-level refactoring often does not suffice, especially in large-scale development.

Spending some time architecting can provide better options in many large-scale development contexts that struggle with applying agile techniques. The cost and benefit tradeoff is often misrepresented as a choice between "do nothing" and "spend a lot of time on something you may not need." The concrete benefit of having real options requires the tradeoff to be made between "do nothing, possibly suffer a lot later" and "do just a little, suffer less later."

Identifying architectural elements that enable future stakeholder goals requires mapping options to releases across the lifespan of the system. A real option often requires some portion of the system to be developed today to enable future development at ease. Understanding which release that option needs to be allocated to and how its cost will be paid during that release are key to success. The release planning board provides a visual means to monitor such elements throughout the releases. Although lower in cost, options are not without expense, so there should not be too many. But cost is not the only issue, so a large-scale project without any options should be viewed with a critical eye. Ideally, the decision to develop an option should be justified by the desire to mitigate the risk of an uncertain future.

Identifying dependencies within a given release also requires understanding the deliberate shortcuts taken to achieve the high-priority functionality. These shortcuts (technical debt) need to be revisited at each iteration. Monitoring these decisions is the first step to realizing the good enough, but cost effective solution today without endangering the needed full solution tomorrow. Once identified, the decision can be made at appropriate times to emphasize more architecting and paying off the debt as opposed to adding new features.

Looking back at the conceptual architecture shown in Figure 3, even at this level, several decisions can be made by taking advantage of dependency analysis in relationship to real options and technical debt concepts. The App catalog management capability describes the feature allowing users to author and add apps to the app store. The matrix shows that the Self-publishing component has a role in implementing this feature. Depending on the cost and value of early delivery versus the level of control, two approaches are available. In a quick delivery approach, rather than implement the full functionality in a separate Self-publishing component, initially a subset could be implemented in the Store administration component that has been selected for implementation in an early release for other reasons. Administrator users have full access to this component through the Sales management capability. This approach would depend on the administrator to ensure that only authorized and well-behaved apps are published, but since this approach limits exposure of the infrastructure and is simpler to implement, it could be deployed quicker. In conjunction with this approach, preparing for the future release and creating the infrastructure for self publishing can be an option for future investment. When the time comes, the infrastructure could be self enabled, increasing the innovation of apps by allowing users to submit their own without external controls.

Technical debt is most often associated at the level of detailed design and code artifacts and tool support is beginning to show promise [9]. An analog for monitoring and managing technical debt in the architecture would provide analyses earlier in the development cycle for keeping the project on track. Some of these measures exist and can be used today. For example, Hinsman from L.L. Bean [10] used a tool to analyze and monitor architecture violations based on dependency analysis in an ongoing effort to evolve and improve its architecture. Once the architecture was restructured, the process was modified to support agility through keeping the architectural elements visible so that they could be explicitly managed.

Key Take-Aways

A focus on architecture is not in opposition to Agile values and principles. In fact, ongoing sustainable achievement of Enhancement Agility is only possible when coupled with Architectural Agility. To achieve Architectural Agility, the Agile community must first expand its focus on end user stories and address the broader topic of capabilities, including quality attribute requirements and a diverse range of stakeholders. The use of dependency analysis practices can be used to facilitate a "just-in-time" approach to building out the architectural infrastructure. Real options and technical debt heuristics can be used to optimize architectural investment decisions by analyzing uncertainty and tradeoffs between incurred cost and anticipated value.

ABOUT THE AUTHORS

The authors work in the Research, Technology, and System Solutions Program at the Software Engineering Institute and are currently engaged in a research project on "Communicating the Value of Architecting within Agile Development."

Nanette Brown is a Visiting Scientist and is a Principal Consultant with NoteWell Consulting. She is engaged in activities focusing on architecture within an Agile context. Previously, Nanette worked at Pitney Bowes Inc., most recently as Director of Architecture and Quality Management, where she was responsible for design and implementation of a customized SDLC that blended RUP and Agile practices.

Robert L. Nord is a senior member of the technical staff and works to develop and communicate effective methods and practices for software architecture. He is co-author of the practitioner oriented books, Applied Software Architecture and Documenting Software Architectures: Views and Beyond, published by Addison-Wesley and lectures on architecture-centric approaches.

Ipek Ozkaya is a senior member of the technical staff. Her primary work is on developing techniques and methods for improving software architecture practices by focusing on software economics and requirements management. Currently, she serves as the technical lead of the agile development and software architecture independent research work, in addition to leading work in architecture-based system evolution. In addition she contributes to teaching of several courses in the Software Architecture Certificate Program at the SEI.

Contact Information

Nanette Brown
nb@sei.cmu.edu

Robert L. Nord
rn@sei.cmu.edu

Ipek Ozkaya
ozkaya@sei.cmu.edu

Software Engineering Institute

Carnegie Mellon University

4500 Fifth Avenue
Pittsburgh, PA 15213-3890
Tel: +1 (412) 268-7700
Fax: +1 (412) 268-5758
URL: http://www.sei.cmu.edu/architecture/

REFERENCES

1. Lemnios, Z. (2010) Statement of Testimony Before the United States House of Representatives Committee on Armed Services Subcommittee on Terrorism, Unconventional Threats and Capabilities, March 23, 2010. [cited on June 11, 2010] URL: <http://www.dod.mil/ddre/Mar232010Lemnios.pdf>
2. Cohan, Sean (2007) *Successful Integration of Agile Development Techniques within DISA*, AGILE 2007.
3. Crowe, P, Cloutier, R. (2009) "Evolutionary Capabilities Developed and Fielded in Nine Months," *CrossTalk*, May 2009. URL: <http://www.stsc.hill.af.mil/crosstalk/2009/05/0905CroweCloutier.html>
4. Denne, M., & Cleland-Huang, J. *Software by Numbers: Low-Risk, High-Return Development*. Upper Saddle River, N.J.: Prentice Hall. 2004.
5. CIO/G-6 Public Affairs. G-6 launches 'Apps for the Army' challenge. [cited on June 11, 2010] URL: <http://www.army.mil/-news/2010/03/01/35148-g-6-launches-apps-for-the-army-challenge/>
6. Carriere, J. Kazman, R., Ozkaya, I. "A Cost-Benefit Framework for Making Architectural Decisions in a Business Context" in Proceedings of the 32nd International Conference on Software Engineering, Vol 2, pp:149-157, 2010
7. Bahsoon, R., Emmerich, W., Macke, J. "Using Real Options to Select Stable Middleware-Induced Software Architectures." *IEE Proceedings Software* - Special issue on relating software requirements to architectures 152(4) (2005) ISSN 1462-5970, pp. 153-167, IEE press.
8. Fowler, M. *Technical Debt Quadrant*. Bliki [Blog] 2009 [cited on June 14, 2010]; URL: <http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html>
9. Gaudin, O. Evaluate your technical debt with sonar, [cited on June 11, 2010] URL: <http://www.sonarsource.org/evaluate-your-technical-debt-with-sonar>
10. Hinsman C., Sangal, N., Stafford, J. *Achieving Agility Through Architecture Visibility*, in LNCS 5581/2009, Architectures for Adaptive Software Systems, 2009 pp.116-129

Figure 1: A typical software-reliant system: the M1 Abrams tank relies on software for navigation, targeting, precision fires, and more.[1]

The Chief Software Architect in U.S. Army Acquisition

Stephen Blanchette, Jr. and John Bergey
Software Engineering Institute

Abstract: The U.S. Army is aggressively pursuing software architecture practices as a means of reducing risk in its acquisition programs. Central to this strategy is creating an appropriately skilled workforce capable of overseeing software development activities in its innovative programs. The latest development in the Army's long-standing pursuit of improving the software talents of its acquisition workforce is the establishment of Chief Software Architects in its program executive offices. This article discusses this latest demonstration of the Army's commitment to adopting an architecture-centric acquisition approach and its focus on developing the software architecture skills of its acquisition workforce.

In May of 2009, Lieutenant General Ross Thompson, then the military deputy to the assistant secretary of the Army for acquisition, logistics and technology (ASA[ALT]), issued a memorandum directing each Program Executive Office (PEO) to designate a Chief Software Architect (CSWA). The directive was another step in the Army's aggressive efforts to instill architecture-centric practices across its acquisition programs. Since late 2002, the ASA(ALT) has been working with the Carnegie Mellon® Software Engineering Institute (SEI)—a federally funded research and development center—in a strategic partnership known as the Army Strategic Software Improvement Program (ASSIP). The aim of this partnership is to improve the Army's ability to acquire software-reliant systems (Figure 1)—i.e., systems whose behavior (e.g., functionality, performance, safety, security, interoperability, and so forth) is highly dependent on software in some significant way. Through this partnership, the Army is enhancing its ability to be a "smart buyer" of software-reliant systems.

Early ASSIP investigations into Army acquisition programs indicated, among other things, that while software-architecture practices were deemed important for software-reliant systems programs, the methods and skills to carry out those practices were perceived to be inadequate. Hence, the ASSIP formulated an initiative to raise the organic capabilities of the Army acquisition workforce in the area of architecture-centric software development. This article discusses the Army's software architecture initiative and examines the human factor behind the technology: the Chief Software Architect.

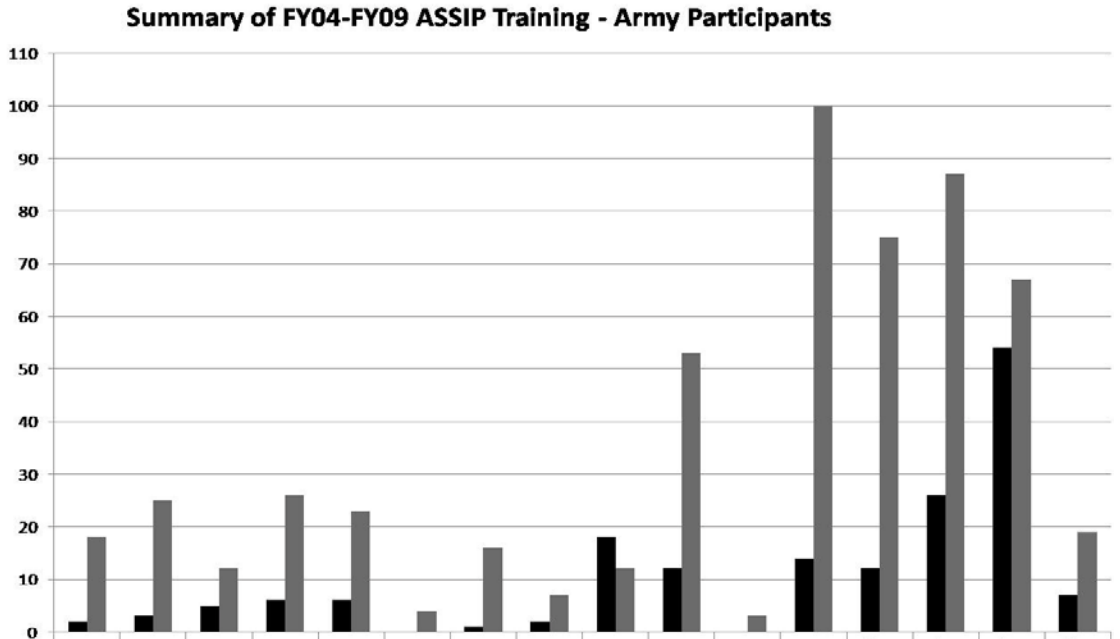
The Importance of Software Architecture

When viewed in terms of program impact, the reason for focusing on software architecture becomes obvious. Experience confirms that the quality and longevity of a software-reliant system is largely determined by its architecture. The software architecture underpins a system's software design and code; it represents the earliest design decisions, ones that are difficult and costly to change later [1]. Further, the software architecture supports, or impedes, the desired system qualities that are manifest in the software, so getting the architecture "right" has enormous implications both for the software and for the parent system that is reliant upon that software to deliver any part of its functionality. The right software architecture will facilitate user acceptance of a system; the wrong one will do quite the opposite. As confirmed by a number of studies in the last decade [2, 3, 4, 5], sound software architectural practices are essential to successful software-reliant systems programs.

However, history has shown that the linkage between software architecture practices and successful acquisition of software-reliant systems has not been sufficient motivation to incorporate such practices in acquisition programs. According to a 2009 NASA study on flight software complexity, "Good software architecture is the most important defense against incidental complexity in software designs, but good architecting skills are not common" [6]. Indeed, reports repeatedly cite poor architectural practices and a general lack of understanding of the need for software architecture as a source of acquisition program difficulties [7, 8, 9, 10, 11].

Thus, while an architecture-centric development approach is an acknowledged good practice in software-reliant systems programs, it is rarely executed effectively or rigorously.

Figure 2: Summary of ASSIP Architecture Training – Army Participants



	Other	PEO 1	PEO 2	PEO 3	PEO 4	PEO 5	PEO 6	PEO 7	PEO 8	PEO 9	PEO 10	PEO 11	Software Center 1	Software Center 2	Software Center 3	Software Center 4
Total Certificate Earners (SAP and/or ATAM Eval)	2	3	5	6	6		1	2	18	12		14	12	26	54	7
Total Participants	18	25	12	26	23	4	16	7	12	53	3	100	75	87	67	19

The ASSIP Software Architecture Initiative

Recognizing that software architecture is still one of the key technical challenge areas facing its Project Management Offices (PMOs), the Army devoted a significant part of its ASSIP resources to address the problem by creating a software architecture initiative. Initially, a training component formed the core of the initiative.

The SEI already had available a formal training curriculum for software architecture,[2] and the ASSIP elected to use it as the basis of the software architecture initiative's training element. The curriculum consists of six courses:

>> Software Architecture: Principles and Practices
>> Documenting Software Architectures
>> Software Architecture Design and Analysis
>> Software Product Lines
>> SEI Architecture Tradeoff Analysis Method® (ATAM®) Evaluator Training
>> ATAM Leader Training

The SEI delivered the curriculum at the Army Software Engineering Centers (SECs) using the same materials and instructors as in its publicly offered classes. The SECs provided the most central location for many participants since most of the Army's PMOs are located in close proximity to one of the SECs. Students who completed the prescribed course sequences earned certificates just as if they had attended the regular public offerings.[3]

The training program enjoyed strong participation, a good indication of both need and interest within the Army acquisition community. In fact, demand exceeded expectations and forced the waving of class size restrictions in a few instances. Additionally, participation was broad, with representation from all 11 PEOs[4] and all of the Army's software centers. Well over 500 Army technical professionals have attended at least part of the curriculum, with more than 25% having earned at least one certificate. Figure 2 summarizes these results.[5,6]

In addition to training practitioners, the ASSIP builds awareness at higher levels: A rotating list of Army senior leaders, personally invited by the MILDEP, gain exposure to software architecture and other important software engineering concepts three times a year during the ASSIP senior leader education program.

Beyond training, the ASSIP software architecture initiative grew to include a skill-building component. The initiative sponsored several ATAM-based software architecture evaluations, with the proviso that trained Army evaluators would participate as evaluation team members. (Projects that had not yet developed a software architecture conducted Quality Attribute Workshops, or QAWs, usually as a precursor to an ATAM evaluation.) Table 1 shows the projects that have participated to date. The evaluations allowed trained Army personnel to practice their skills and also introduced architecture-centric practices across a variety of Army projects.

Table 1: Projects Employing Architecture-Centric Practices

Army Project (in alphabetical order)	ATAM	QAW
Aerial Common Sensor	✓	✓
Army Battle Command System		✓
Command Post of the Future	✓	
Common Avionics Architecture System	✓	
Distributed Common Ground Station – Army	✓	✓
Force XXI Command Brigade-and-Below	✓	
Future Combat Systems	✓	✓
Integrated Fired Control	✓	✓
Joint Tactical Common Operational Picture Workstation	✓	
Manned/Unmanned Common Architecture Program	✓	
Network Operations Data Product Development Environment		✓
One Semi-Automated Forces	✓	
Sequoyah		✓
Warfighter Information Network – Tactical	✓	

According to a recent study, these architecture-centric practices have had a positive impact [12]. As shown in Figure 3, most projects reported significant improvement in their architecturally significant artifacts (including system quality attributes, software architectures themselves, and architecture-related risks). The architecture teams achieved an understanding of stakeholder expectations and the implications of architectural decisions on user needs [12]. Additionally, almost all projects experienced very substantial or significant improvement in stakeholder communication (see Figure 4). Stakeholders, collectively, achieved a common understanding of the systems under development, which increased the likelihood that those systems would address expectations and user needs (and, consequently, improved the chances for program success) [12].

The Role of the Army's CSWAs

Having trained a cadre of acquisition professionals capable of implementing architecture-centric practices, the next step for the Army was to begin the institutionalization of software architecture practices throughout its acquisition offices. LTG Thompson decided that the best way to accomplish that goal was to establish Chief Software Architects in the program executive offices. Each PEO has oversight responsibility for a domain of related projects and products:[7]

>> PEO Ammunition (Ammo)
>> PEO Aviation (AVN)
>> Joint PEO Chemical and Biological Defense (CBD)
>> PEO Combat Support and Combat Service Support (CS&CSS)
>> PEO Command Control and Communications – Tactical (C3T)
>> PEO Enterprise Information Systems (EIS)
>> PEO Ground Combat Systems (GCS)
>> PEO Integration
>> PEO Intelligence, Electronic Warfare and Sensors (IEW&S)
>> PEO Missile and Space (MS)
>> PEO Simulation, Training, and Instrumentation (STRI)
>> PEO Soldier

Each CSWA is responsible for providing guidance for software issues across a PEO's portfolio of programs. The scope of responsibility is broad; the CSWAs are accountable for oversight and management of all software being developed or acquired within their respective PEOs. Consequently, the position requires strong software competence and pertinent training. Particularly notable in the CSWA directive is the specific requirement for training. The intent is that the position is not just another task in someone's job jar; the CSWAs are expected to possess or obtain skills relevant to the position. Each CSWA must complete training equivalent to the SEI course series for Software Architecture Professionals. A subset of the architecture curriculum, the Software Architecture Professional series consists of a foundational course in software architecture principles and practices (including a compulsory competency examination), as well as in-depth courses covering essential concepts for effectively designing and analyzing software architectures, effective documentation methods, and an introduction to software product line concepts. These are advanced topics; the coursework assumes attendees already are practicing software professionals with responsibility for designing, developing, or managing the construction of software-reliant systems.

Figure 3: Architecture-Centric Practices Improve Artifacts

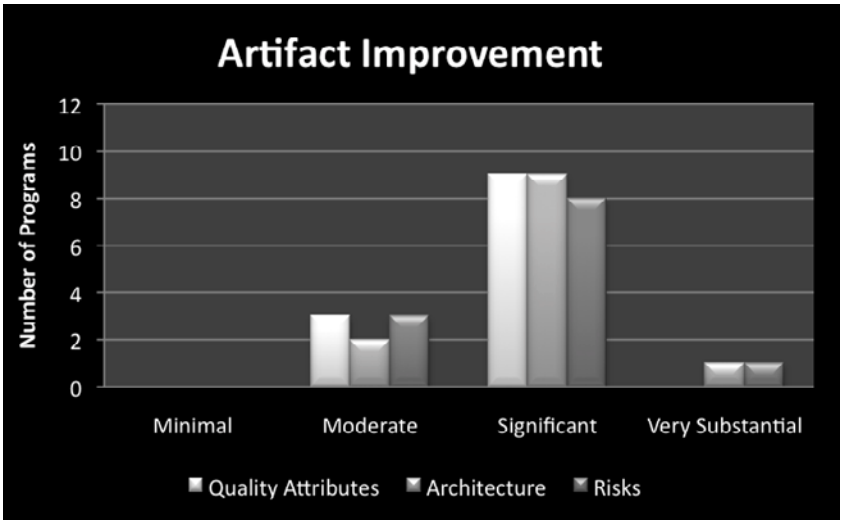
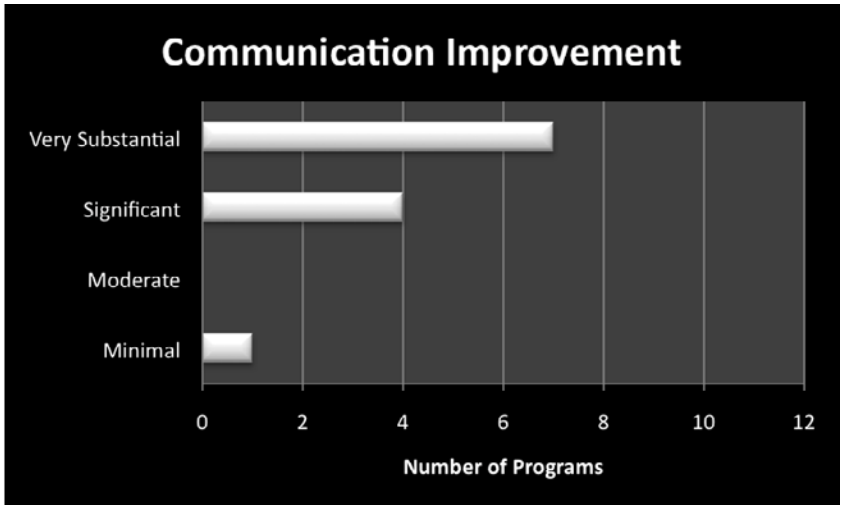


Figure 4: Architecture-Centric Practices Improve Communication



In August 2009, the CSWAs met together for the first time during an ASSIP Action Group meeting. There, they fleshed out their collective responsibilities in more detail. They identified their primary task as providing support to project managers (PMs) with their software processes, including monitoring software architecture development from initial design decisions throughout the acquisition life cycle in order to identify and mitigate software risks, linking architectural components to mission drivers, and focusing on stakeholder requirements. The CSWAs will help ensure every PM has an appropriately documented software architecture and will help to evaluate how well individual systems meet the appropriate quality attributes. Beyond the architecture, the CSWAs will assess and evaluate software cost estimates in a system life cycle context for portfolio programs as well as review and endorse system engineering plans with their respective Chief System Engineers to ensure those plans leverage appropriate standards[8] and appropriate architecture-centric practices.

A second task for the CSWAs is to establish the necessary infrastructures within their PEOs to support software objectives, including issuing guidance to the PMs on software architecture requirements, identifying and enforcing any PEO-specific system quality attributes that will be implemented in software, and providing guidance for software architecture design and reviews to ensure consistent implementation of best practices.

The CSWAs' third task is to decide the best ways to leverage software architecture to mitigate program risks, especially with regard to analysis in response to integration and interoperability challenges. In particular, they will ensure development of software architectures in a system of systems context to address the interoperability requirements that are becoming more common across all Army systems.

Lastly, the CSWAs will participate in the ASSIP and other Army-wide communities of interest to exploit opportunities for commonality across the PEO portfolios.

Way Ahead

The Office of the DoD Chief Information Officer issued a white paper [13] on a competency framework for the DoD Architect that noted three root causes for shortcomings in architecture practices across the DoD:

>> **Inability to leverage the benefits of an architecture due to inadequate training on the part of stakeholders or inadequate communication on the part of architects**

>> **Lack of incentives to encourage the professional growth of architects in the DoD**

>> **Lack of visibility into the existence or value of architecture training**

All the services have made some strides with respect to system-level architecture (the Navy's Open Architecture initiative, for example, instituted relevant policy supported by a model and a corresponding tool [14]). However, through the ASSIP and the CSWAs, the Army has leapt ahead with a comprehensive strategy for software architecture that addresses not just technical issues but also these non-technical aspects, which are essential to institutionalization and achieving maximum benefit from software architecture practices. The goal now is to help ensure that the new Army CSWAs are positioned for success. To that end, the FY10 ASSIP plan focuses on supporting them with continued training and awareness opportunities as well as technical assistance.

In working with the CSWAs to develop execution plans, one non-technical theme recurs: How can a CSWA direct and influence within organizational constraints? Since the CSWAs exercise no direct authority over the projects within their respective PEO portfolios, the question is a crucial one. As a solution, most CSWAs are taking a relationship-building approach, teaming with PMO software architects and engineers to work on problems collaboratively. In so doing, they will be able to leverage early adopters of software architecture practices to achieve initial successes and build publicity within their organizations. In addition, some CSWAs are seeking formal endorsement from their PEOs or Chief System Engineers as a means of putting more weight behind their objectives.

From a technical perspective, feedback from the CSWAs indicated some challenges. One challenge is using software architecture to help understand, validate, and improve software cost estimation. Intuitively, a better understanding of a software architecture should lead to a better understanding of the software to be built, which in turn should lead to a better estimate of software cost. However, CSWAs need tools and methods to formalize the relationship between architecture and cost estimation. Another challenge is developing a standard means of determining appropriate technology readiness levels (TRLs) for software, and determining which phases of the acquisition lifecycle require which software TRLs.

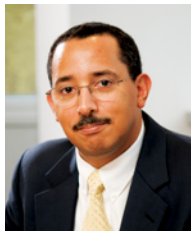
Overall, the positioning of the CSWAs at the PEO level is advantageous in that it enables them to take a portfolio perspective on such important issues, as well as on architecture sub-specialties such as data architecture and security architecture, instead of developing solutions project by project. Data and security architectures, particularly, are vital for implementing robust and reliable networked solutions for the warfighter, and such solutions are becoming increasingly commonplace. Further, and perhaps more importantly, the CSWAs are able to collaborate with each other through the ASSIP forum to address these software architecture matters at the system of systems level, which will facilitate the development of truly interoperable capabilities for a modernized Army and for joint and coalition forces.

Summary

The creation of a Chief Software Architect role in each PEO has been a significant step in the Army's efforts to institutionalize architecture-centric practices in its software-reliant system acquisition programs. Through the ASSIP, the Army has focused on developing the software architecture skills of its acquisition workforce and building awareness of architecture-centric practices among its leadership. The CSWAs can leverage the cadre of software architecture professionals and qualified ATAM evaluators to realize the benefits of architecture-centric practices across the Army's acquisition projects and set the standard for improvement across the DoD.

The ASSIP continues to support the CSWAs as they work to establish and champion architecture-centric practices within their PEOs.

ABOUT THE AUTHORS



Stephen Blanchette, Jr. is Chief Engineer for Army Programs at SEI. He has more than 23 years experience in the defense industry as a software engineer and manager. He is an associate fellow of the American Institute of Aeronautics and Astronautics and a senior member of the Institute of Electrical and Electronics Engineers. Mr. Blanchette earned a BS in Computer Science from Embry-Riddle Aeronautical University and an MA in Diplomacy from Norwich University.
E-mail: sblanche@sei.cmu.edu

John Bergey is a senior member of the technical staff at SEI, specializing in transitioning SEI product line and architecture-centric practices (e.g., Quality Attribute Workshop, Architecture Tradeoff Analysis Method®) into acquisition practice across the Armed Services. Prior to joining SEI, he served over 25 years as a software division manager with the U.S. Navy. Mr. Bergey is a graduate of Pennsylvania State University and the University of Michigan.
E-mail: jkb@sei.cmu.edu

Software Engineering Institute
4500 Fifth Avenue
Pittsburgh, PA 15213

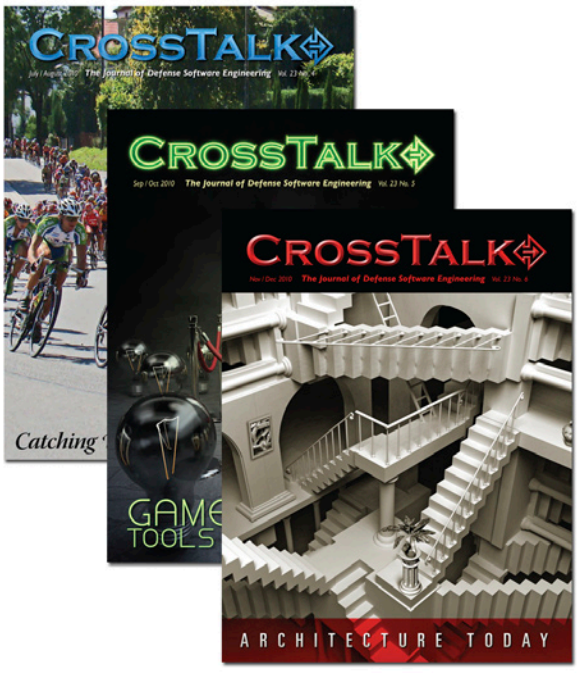
REFERENCES

1. Bass, Len, Paul Clements, and Rick Kazman. *Software Architecture in Practice, (2nd ed.)*. Boston: Addison-Wesley, 2003.
2. Defense Science Board Task Force. "Acquiring Defense Software Commercially." June 1994. 25 December 2009 <http://www.acq.osd.mil/dsb/reports/commrecialdefensesoftware.pdf>.
3. ---. "Defense Software." November 2000. 25 December 2009 <http://www.acq.osd.mil/dsb/reports/defensesoftware.pdf>.
4. Anderson, William, et al. *Army workshop on lessons learned from software upgrade programs (CMU/SEI-2001-SR-021)*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
5. Government Accountability Office. Defense Business Transformation, Sustaining Progress Requires Continuity of Leadership and an Integrated Approach (GAO-08-462T). Washington: Government Accountability Office, 2008.
6. Dvorak, Daniel L. "NASA Study on Flight Software Complexity." 2009.
7. Charette, Robert, John McGarry and Kristen Baldwin. *Tri-service Assessment Initiative Phase 2 Systemic Analysis Results*. January 2003.
8. Bass, Len, Robert Nord and William G Wood. "Risk Themes from ATAM Data: Preliminary Results." April 2006. 25 December 2005 <http://www.sei.cmu.edu/library/abstracts/presentations/basssaturn.cfm>.
9. Kasunic, Mark. *Army Strategic Software Improvement Program (ASSIP) Survey of Army Acquisition Managers (CMU/SEI-2004-TR-003)*. Pittsburgh, PA: Software Engineering Institute/Carnegie Mellon University, 2004.
10. Keeler, Kristi L. *U.S. Army Acquisition – The Program Office Perspective (CMU/SEI-2005-SR-014)*. Pittsburgh, PA: Software Engineering Institute/Carnegie Mellon University, 2005.
11. Blanchette, Jr., Stephen. *U.S. Army Acquisition – the Program Executive Officer Perspective (CMU/SEI-2005-SR-002)*. Pittsburgh, PA: Software Engineering Institute/Carnegie Mellon University, 2005.
12. Nord, Robert L., John Bergey, Stephen Blanchette, Jr., and Mark Klein. *Impact of Army Architecture Evaluations (CMU/SEI-2009-SR-007)*. Pittsburgh, PA: Software Engineering Insttitue/Carnegie Mellon University, 2009.
13. Office of the DoD CIO. White Paper Phase I: A Competency Framework for the DoD Architect. Washington: Department of Defense, 2008.
14. Shannon, Jim. "Naval Enterprise Open Architecture: What Program Managers Need to Know." January 2006.

REFERENCES (continued)

NOTES

1. DoD photo By Staff Sgt. Aaron Allmon, U.S. Air Force. (Released). Retrieved 12/23/2009 from <http://www.defense.gov/dodcmsshare/newsphoto/2006-02/060203-F-7823A-008.jpg>.
2. Interested readers will find additional information about the SEI software architecture curriculum on the SEI website: <http://www.sei.cmu.edu/training/find/architecture.cfm>.
3. Three certificates, Software Architecture Professional, ATAM Evaluator, and ATAM Lead Evaluator, are available to students who complete the required courses. Beginning in 2009, individuals seeking one of these certificates were required to pass a validation exam in addition to completing the coursework.
4. Data for PEO Missiles and Space include its predecessor organizations PEO Tactical Missiles and PEO Air Space and Missile Defense; PEO Integration, created in mid-2009, is not represented in the data.
5. PEOs and software centers are shown in random order.
6. In addition to Army personnel, 62 representatives from other services and support contractors have been trained through the conclusion of FY09.
7. In addition to the Army PEOs noted in the list, the Joint PEO for the Joint Tactical Radio System (JTRS), currently transitioning from the Navy to the Army, also will participate.
8. Examples of such standards are International Standards Organization/International Electrotechnical Commission (ISO/IEC) standards 15288 for system engineering and 12207 for software development.



CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:

People Solutions to Software Problems
May/June 2011
Submission Deadline: December 10, 2010

DoD Gaming and Virtual World Applications
July/August 2011
Submission Deadline: February 11, 2011

Protecting Against Predatory Practices
September/October 2011
Submission Deadline: April 8, 2011

Please follow the Author Guidelines for CROSSTALK, available on the Internet at <www.crosstalkonline.org/submission-guidelines>. We accept article submissions on software-related topics at any time, along with Letters to the Editor and BackTalk. To see a list of themes for upcoming issues or to learn more about the types of articles we're looking for visit <www.crosstalkonline.org/theme-calendar>.

XDDS:

A Scalable Guard-Agnostic Cross Domain Discovery Service

Michael Atighetchi, Raytheon BBN Technologies
Joseph Loyall, Raytheon BBN Technologies
Jonathan Webb, Raytheon BBN Technologies
Michael J. Mayhew, AFRL/RIEB

Abstract: As both the DoD and the Intelligence Community (IC) are moving toward service-oriented architecture (SOA), it is important to ensure that SOA-based systems can operate and exchange classified information across domain boundaries in support of net-centric missions. The interplay between SOA and cross domain solutions (CDS) raises a number of challenges that are grounded in the inherent mismatch between core SOA principles, such as loose coupling, composability, and discoverability, and current CDS technologies and certification and accreditation processes in use today. The Cross Domain Discovery Service (XDDS) described in this paper provides an architecture and design for extending service discovery, a core SOA functionality, across domain boundaries. The resulting services and protocols provide access to service information across security domains in a secure, guard-agnostic, scalable, and flexible way that is amenable to certification and accreditation (C&A).

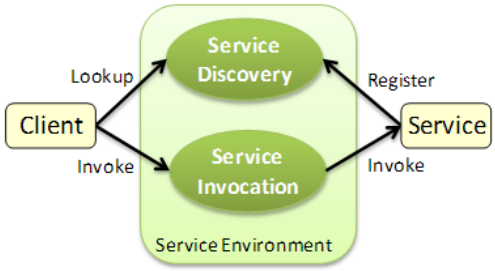
Introduction

SOA is becoming increasingly important to, and entrenched in, the DoD and IC for military and intelligence operations, including initiatives such as Net-Centric Enterprise Services (NCES). While SOA includes services and support for security, such as access control, these initiatives have largely concentrated only on providing security within domains,[1] not across them. Simultaneously, CDS[2] have begun to handle the growing requirement to service the need to share information critical to military operations, disaster response, national intelligence, and other situations, carefully balancing the need to share with the traditional need to protect sensitive or classified information within and across domains.

Discovery services play an important role in single domain SOAs because of the dynamic nature of a service environment. As services become available, change, or get removed, applications need to have up-to-date information about the definition of available services. Management of static depictions of these environments becomes difficult, both within and across domains, particularly as the number of services increases. This motivates a requirement for discovery services across domains that is currently unmet by existing service discovery solutions, which only work within domains.

Discovery itself is a simple process, as shown in Fig. 1. A service registers itself with the service discovery service that is part of an existing service environment. Next, a client (shown on the left) performs lookup requests on the service discovery service to find newly registered services. Once the client has found a suitable service, it proceeds to invoke that service through a specific invocation mechanism.

Fig. 1. Functional View of the Discovery Process within a Single Domain



The XDDS described in this paper fills this gap by enabling dynamic discovery and use of services across a variety of domains and associated relationships, including hierarchical, non-hierarchical, and coalition. The resulting services and protocols provide access to service information across security domains in a secure, guard-agnostic, scalable, and flexible way that is amenable to C&A following standard IC and DoD processes, e.g., DIACAP [1], NIST Special Publication 800-53 [2], or ICD 503 [3]. The XDDS prototype addresses requirements expected of any new cross domain capability in an early research and development prototype lifecycle. The XDDS prototype is:

- >> Guard-agnostic, i.e., independent of any specific guard implementation.
- >> Modular, enabling reuse of existing guards and services that have successfully passed C&A
- >> Developed, documented, and tested with C&A in mind and an explicit goal to provide a body of evidence for certification and accreditation processes in later phases.

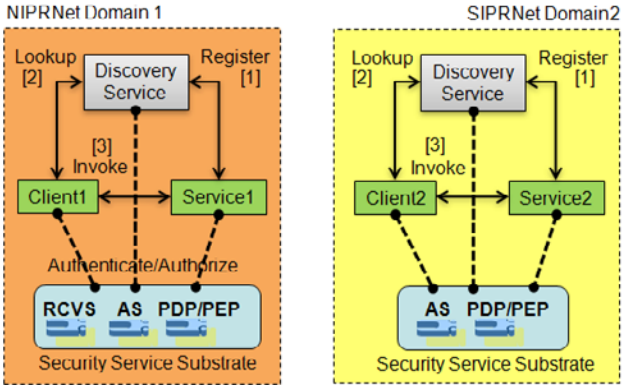
DoD-Centric Use Case

Current support for net-centric operations is based on isolated deployments of relevant services in individual domains. Fig. 2 illustrates a NIPRNet and SIPRNet deployment of the DISA NCES service discovery service, which provides the ability to register services (in step 1), lookup services

(step 2), and finally invoke services (step 3), but only within a respective domain. Extending the discovery of services across domains necessitates introduction of new cross flows for either disseminating service registrations or lookup requests. The flows also need to contain filters to ensure that clients only get access to information they are entitled to, even from remote domains.

As part of the XDDS effort described in this paper, we designed and prototyped services and cross domain protocols that enable Client1 in Fig. 2 to discover and use Service2, and conversely Client2 to discover and use Service1, if and only if these interactions are permissible under existing cross domain data sharing policies.

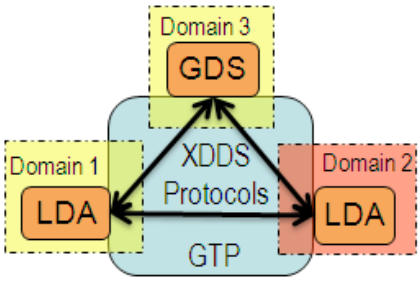
Fig. 2. Current Service Discovery in DoD Enterprise Environments



XDDS Architecture

We created an extensible and flexible architecture for cross domain service discovery and implemented versions of the following components shown in Fig. 3.

Fig. 3. XDDS Architecture



A Local Discovery Agent (LDA) in each domain transparently intercepts lookup and registration requests within domains and handles routing requests across domains for performing local discovery, e.g., using Universal Description Discovery and Integration (UDDI). The LDAs enable transparent discovery across domain boundaries without requiring code changes to existing discovery services or clients.

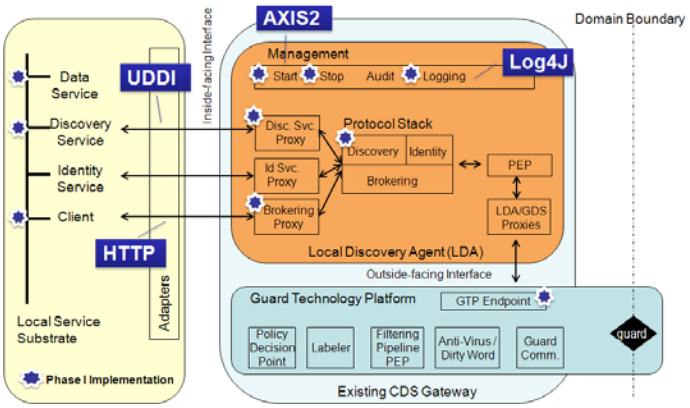
The XDDS message protocol, through which LDAs interact with one another, is based on two simple generalized communication models, namely referral- and replication-based discovery, and accommodates requirements on message exchanges in cross domain environments, such as restricted XML schemas. Our protocol can encapsulate a large number of variant discovery protocols without significant changes, minimizing the impact of changes on C&A.

The Global Discovery Service (GDS) is an extended LDA component that facilitates scalability by introducing hierarchy through which any number of LDAs can interact. The GDS maintains information about the domains and how they can reach one another, enforces policies on cross domain interactions, facilitates proper authentication, and supports anonymization of domains.

The Guard Technology Platform (GTP) is a generic interface to existing guards supporting the examination of cross domain flows represented with the XDDS protocol. Using the GTP abstraction during development allows XDDS to remain independent of specific guard and CDS implementations.

Fig. 4 shows the XDDS architecture in more detail. The LDA is strategically placed between the local service substrate on the left, e.g., an ESB, and the guard, which is located closest to the cross domain boundary on the right. Interfaces of the LDA to other components can be categorized into inside-facing and outside-facing. While the inside-facing interfaces talk to existing services in the local domain through adapters, the outside-facing interface interacts with the local endpoint of the Guard Technology Platform within an existing CDS gateway such as the Collaboration Gateway [4] or the Web Service Gateway [5] of the Cross Domain Collaborative Information Environment [6].

Fig. 4. The LDA and its Interfaces to Local Service Substrates and CDS Gateways



For communicating with the local service substrate, the LDA instantiates service proxies and uses adapters for supporting a number of different protocols. In Phase I, we implemented proxies for the discovery and brokering and implemented adapters for UDDI v2 and HTTP.

For communication with other LDAs through the outside-facing interface, the LDA uses LDA service proxies. Since the Phase I prototype only involved two LDAs, we directly linked the LDA with the GTP endpoint through the Simple Object

IMPORTANT STANDARDS AND GUIDANCE DOCUMENTS

- NIST Special Publication 800-95, Guide to Secure Web Services, 8/2007
- MITRE Technical Report MTR080027, Recommendations on the Use of SOAP in a Cross Domain Environment, 2/2008
- MITRE Technical Report MTR040000092, Security Guards for the Future Web, 9/2004
- NSA Report XML Schema Guidance for Cross Domain Security Policy Enforcement, 07/2006
- Intelligence Community Standard for Information Security Marking Metadata (IC ISM), ICS 2007-500-2, Version 2, April 2004
- SOAP 1.1 (W3C Note 08 May 2000) and SOAP 1.2 (W3C Recommendation 27 April 2007)
- WSDL 1.1 Specification (W3C TR Note 14 March 2001)
- OASIS UDDI Technical Note Using WSDL in a UDDI Registry, Version 2.0.2
- OASIS UDDI V2 specification

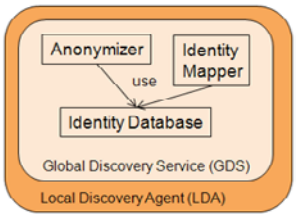
Access Protocol (SOAP). The LDA process itself is implemented as a web service and hosted in the Axis2 [7] web services container. Logging is performed using the Log4J [8] framework.

The heart of the LDA is its layered protocol stack, with a brokering protocol at the lowest layer and discovery and identity management at higher layers. The main purpose of this stack is to convert messages from the local service substrate, which may be complex and technology specific, into a core set of simple messages, which are suited for crossing domain boundaries. One way to describe the collection of core messages is via the notion of abstract protocols for discovery, identity, and brokering. The brokering protocol is responsible for routing requests through the network of XDDS nodes. Requests can either be discovery or identity management related, or originate from a brokering proxy which allows clients to invoke services in other XDDS-enabled domains. Discovery is implemented via exchange of simple messages that allow for registration of local services with XDDS and lookup of registered services throughout the overlay network of XDDS nodes. The identity protocol enables an LDA to export a selected set of identity mappings from the local identity service to other XDDS nodes (such as the GDS).

The LDA contains a Policy Enforcement Point (shown as PEP in Fig. 4) that intercepts requests and subjects them to policy evaluation. In future versions of the LDA, we expect to configure an existing Policy Decision Point with role-based access control policies that determine what information is allowed to be passed outbound and received inbound. To meet the requirement for preventing data leakage between domains, the policy enforcement of high domains always happens on the high side, allowing domains to stay in full control of their data. In addition, low domains implement a second line of defense by pushing protection requirements closer to the source of misbehavior in cases of errors or attacks that are mounted to escalate from low to high domains.

The GDS is built on the same technology platform as the LDA to provide support for anonymization, identity mapping, and LDA synchronization (as displayed in Fig. 5). The discovery service in a GDS operates at a higher layer in the discovery hierarchy in that it manages LDA memberships and allows them to discover each other. LDAs defer to the GDS for discovery requests that they cannot handle and answer discovery queries from the GDS. For identity management, the GDS supports mapping of identities across domains in a scalable way. It can also store identity relevant meta-information about LDAs, such as what identity protocols are supported by an LDA and whether the LDA allows remote verification of identities. For anonymization, the GDS supports multiple operational modes, ranging from traditional onion-routing to support for services that want to disclose only a small subset of information about themselves and implement "don't call us, we'll call you" policies. The GDS allows XDDS to support service discovery even in the most restrictive environments in which the knowledge that a certain domain hosts a certain service is not permitted to cross domain boundaries.

Fig. 5. The GDS



Cross Domain Service Discovery In Action

To ensure feasibility of the XDDS architecture and design and construct a body of evidence for later C&A activities, we implemented a proof-of-concept prototype during Phase I based on the jUDDI open-source server [9].

We started by constructing a baseline scenario for intra-domain discovery of Web Services Description Language (WSDL)-described web services following the WSDL in UDDI OASIS recommendation [10]. We then proceeded to implement referral-based discovery across two domains. Key components of the prototype include an implementation of the XDDS protocol specification together with a set of configurable transformations on UDDI and WSDL documents necessary for cross domain discovery.

The set of transformations, implemented using XSLT, includes scripts to change service end point information, e.g., for making cross domain service calls via existing cross domain web service invocation substrates, as well as to restrict information sharing due to security restrictions, e.g., by redacting UDDI operator identities. The prototype allows flexible control over content and location of transformations applied to the message stream and also rejects messages that do not conform to expectations, e.g., by analyzing sequence numbers to prevent replay attacks.

Fig. 6. Proof-of-Concept Prototype Demonstration

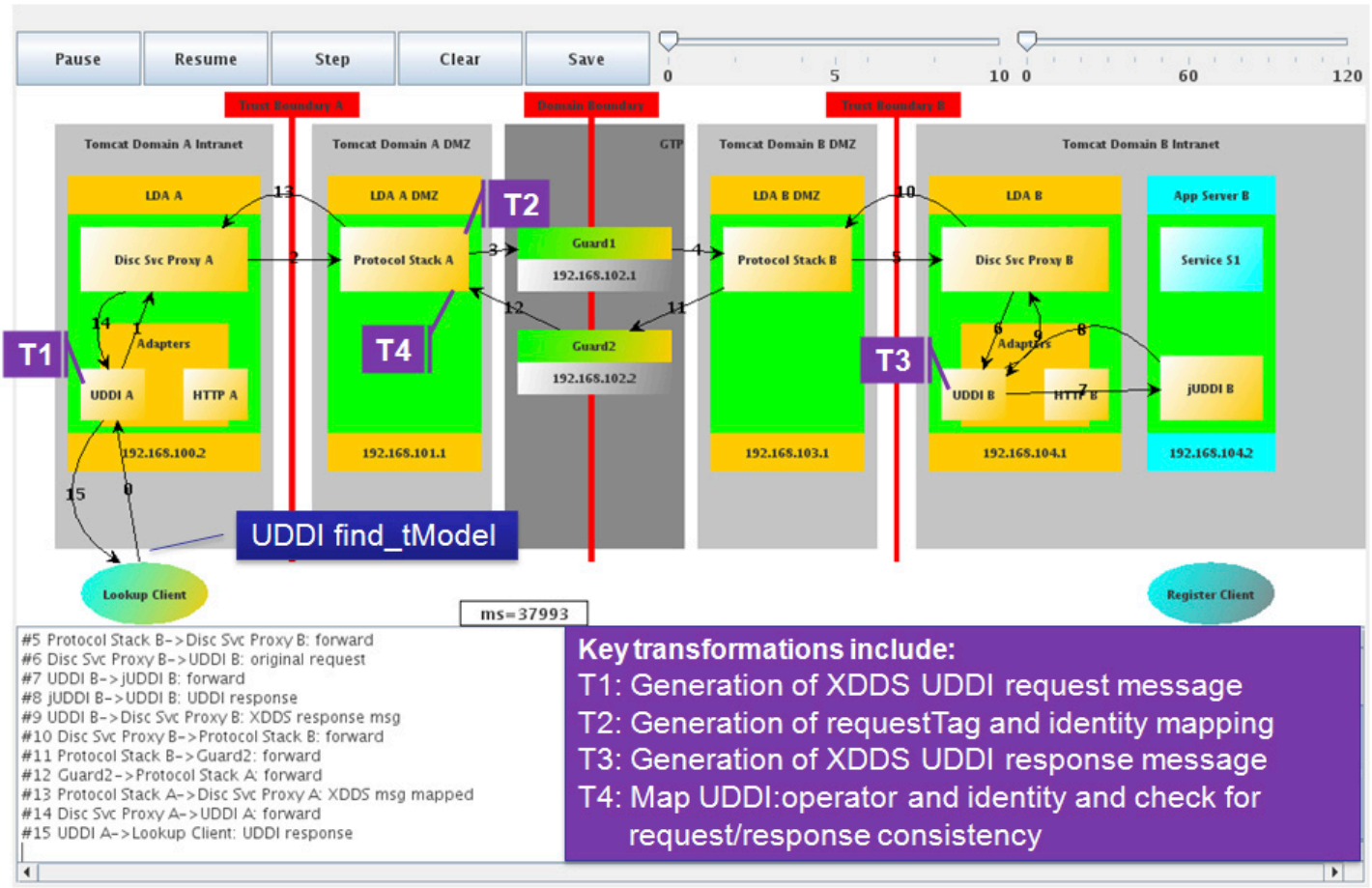


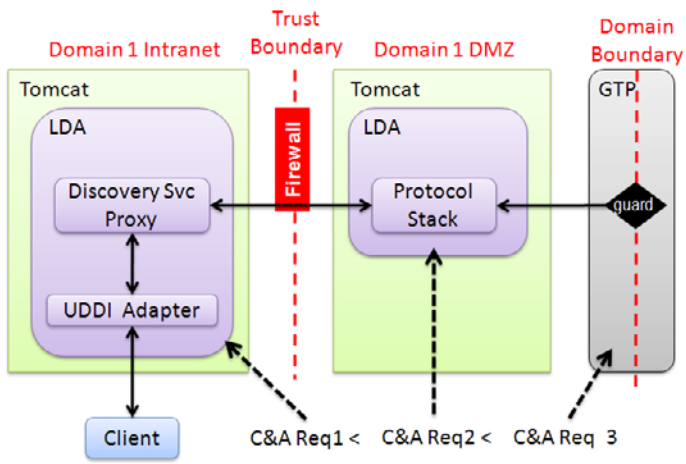
Fig. 6 shows a visualization of the multi-step cross domain discovery process generated from live outputs and logs of participating components. The domain boundary is shown in the center and the GTP is represented through a dark gray box. LDA components are further divided into an intra-net resident LDA process, e.g., LDA A, and a process resident in a Demilitarized Zone[3] (DMZ), e.g., LDA A DMZ. The lookup client is represented by an oval on the left, while the UDDI server is represented by an orange box labeled "jUDDI B" on the right. Fig. 6 shows the sequence of XML message exchanges between various components during a UDDI find_tModel request[4] together with key transformations on the resulting XDDS messages called out via T1 through T4.

Certification and Accreditation of Different Configurations

C&A of CDSs is of significant cost and solutions that do not account for the specifics of cross domain environments will face significant barriers during accreditation. This is even more true for service discovery due to a high degree of technology diversity and proliferation of evolving discovery standards. To address these issues, XDDS decouples existing discovery technologies found locally in a domain from the messages that cross the domain boundary.

The design of the Phase I prototype confines most of the complexity to the protocol adapter, specialized for UDDI in this case, and the discovery service proxy while allowing the LDA/GDS components to exchange a small set of core XDDS XML messages within a narrowly defined message format over the domain boundary (the right side of Fig. 7). Message exchanges across domain boundaries are represented via two generalized communication models, referral and replication (described in more detail later in this section), that cover a wide variety of discovery protocols through adapters.

Fig. 7. Separation of LDA Components along Trust Boundaries



The scope of C&A in this effort was to construct an initial body of evidence that can be used later as the basis for security arguments for a real C&A activity. The various design tradeoffs, use cases, and XML message exchanges and transformations shown through the proof-of-concept prototype all feed into construction of this body of evidence. In addition, we developed the design and proof-of-concept prototype to be consistent with a number of important community documents and standards.

Functional Use Cases and Generalized Communication Models

XDDS supports two basic discovery patterns: referral, where a client request is transferred from a local proxy to a discovery service instance holding the relevant service registration, and replication, where service registrations are copied to local discovery service instances to satisfy local discovery requests.

Basic Discovery Interaction Patterns

The Phase I proof-of-concept prototype supports referral-based discovery, in which the LDA components disseminate lookup requests and corresponding responses across domain boundaries, as shown in Fig. 8.

The sequence of events is as follows:

1) A service in Domain 2 makes a registration request with its local LDA 2. Service description information is only persisted locally.

2) A client in Domain 1 makes a lookup request with its local LDA 1.

3, 4) The LDA 1 forwards the request to LDA 2 in the other domain and receives the response back from LDA 2, which it in turn returns to the client. The transfer of cross domain requests and response is mediated by the GTP.

Fig. 9 depicts the replication-based discovery configuration, and the sequence of steps is as follows:

1) A service in Domain 2 makes a registration request with its local LDA 2.

2, 3) The LDA 2 makes a replication request through the GTP to an affiliated LDA in another domain. Transfer is mediated by the GTP.

4) The replication request is received by the affiliated LDA 1 and any local client requests are serviced by the LDA 1 local to the client C.

Note that a GDS (in its own domain) may be inserted into the communication path to reduce or eliminate the need for multiple point-to-point connections. XDDS provides mixed

Fig. 8. Referral-based Discovery

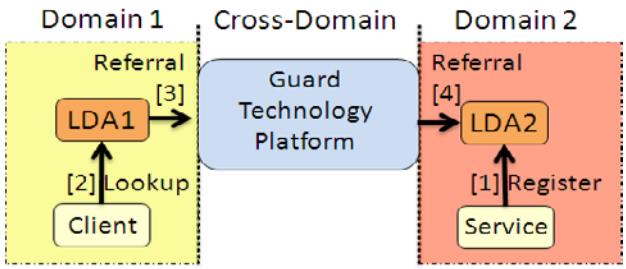
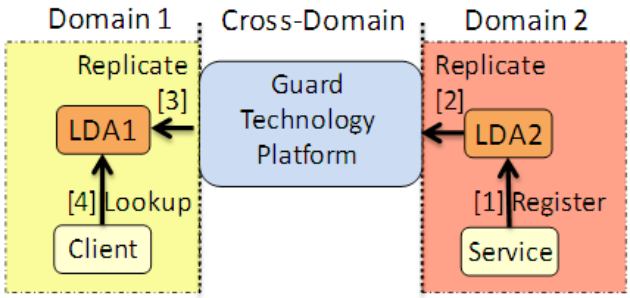


Fig. 9. Replication-based Discovery



operations in which one LDA is configured to replicate registrations to the GDS, while another LDA uses referrals for lookup operations. In both configurations, XDDS carries all cross domain message exchanges over a generic discovery service protocol.

There are two key distinctions between the referral and replication models:

For referral, the information traverses the domain boundary at the time the lookup request is made by the recipient client. For replication, the information traverses the domain boundary at the time the service registration is performed.

In the referral model, the service description is not in the persistent storage of the discovery service element of the requesting domain. The replication model has a persistent copy of the discovery data in all replication domains.

The differences have important implications on security aspects of deployments. For example, it may be more appropriate to replicate service registrations from low to high domains. In this configuration, lookup requests performed in the high domain are handled locally, reducing the amount of risk for interference or covert channels.

XDDS Protocol Specification

The XDDS message protocol is an XML-based message specification that describes the syntax of messages passed between LDAs through the GTP. The protocol is consistent with open standards, e.g., XML, UDDI, Security Assertion Markup Language, WS-Security, XML Signature, and SOAP. The protocol represents XML message exchanges through two basic message forms—XDDS requests (example shown in Fig. 10) and XDDS responses. By default, all requests generate responses and generic acknowledgement responses are returned in error cases instead of error responses. Messages include control information, such as LDA identities used for routing purposes, classification markings, and message integrity and provenance trails that allow enforcement of integrity and anti-spoofing.

Fig. 10. Example XDDS message

```
<xdds:xdds xmlns="http://xdomain.bbn.com/xdds/"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:uddi="urn:uddi-org:api_v2"
 xmlns:wsdl="http://schemas.xml.org/wsdl/" xmlns:xdds="http://xdomain.bbn.com/xdds/">
 <classificationMarking classification="U"/>
 <requestLabel>
  <domainIdentity>DomainA</domainIdentity>
  <requestTag>req_0</requestTag>
 </requestLabel>
 <responderIdentity>
  <domainIdentity>DomainB</domainIdentity>
 </responderIdentity>
 <discoveryQuery>
  <uddiV2Query>
   <find_tModel generic="2.0" xmlns="urn:uddi-org:api_v2">
    <name xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"/>
    <categoryBag xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
     <keyedReference keyName="WSDL type" keyValue="binding" tModelKey="uuid:6e090afa-33e5-
    36eb-81b7-1ca18373f457"/>
      <keyedReference keyName="binding namespace"
       keyValue="http://quickstart.samples/" tModelKey="uuid:d01987d1-ab2e-3013-9be2-
    2a66eb99d824"/>
     </categoryBag>
    </find_tModel>
  </uddiV2Query>
 </discoveryQuery>
</xdds:xdds>
```

To simplify messaging formats, the protocol uses the same message types during referral and replication modes and treats the replication request analogous to a query response in the referral mode. Furthermore, application specific discovery protocols, e.g., UDDI and HTTP, are encapsulated in the XDDS messages in restricted form, allowing the same XDDS message structure to be used with multiple application specific protocols.

The XDDS protocol allows expression of restrictions on message exchanges through both XML schema and XSLT restrictions. The schema restrictions include sanitization of the input stream through removal of non-printing characters and any characters outside the range 040 to 176. In addition, we use white space normalization and disallow any CDATA, Base64, or other similar binary encodings. The protocol handlers further restrict attribute values to enumeration constants or highly constrained value sets and disallow mixed element content. Extensible Stylesheet Language Transformations (XSLT) restrictions are generated automatically from configuration data and tie allowable message exchanges to accredited cross domain flows. For instance, the XDDS protocol handlers use XSLT script to check message ordering and detect message replay scenarios.

Summary and Next Steps

The XDDS Phase I project was a successful research effort that produced significant improvements in technology in a short amount of time. The technology innovations and the XDDS prototype demonstrated in this project are foundational results enabling a necessary capability, previously unavailable, if SOA is to be realizable in DoD and Intelligence Community environments, namely the ability to discover and broker services across domain boundaries in a scalable, safe, and certifiable manner.

In summary, we designed a guard-agnostic architecture for cross domain service discovery based on the principles of modularity, interoperability, transparency, scalability, and security, and produced technical designs for its major components, namely the LDA, the GDS, and the XDDS protocol specification. In addition, we developed use cases involving generalized communication models, namely referral-based and replication-based discovery, advanced discovery capabilities, including hierarchical and anonymous discovery processes, and assured discovery capabilities through authentication and authorization, message protection through signatures, and traffic restrictions and normalization. Finally, we successfully demonstrated cross domain discovery via a proof-of-concept implementation of one specific configuration supported by the design.

Our plans for future work include expansion of the existing proof-of-concept capabilities by a) adding replication-based discovery and initial authentication, authorization, and service invocation capabilities, b) developing the first version of the GDS component to enable hierarchical discovery and enhance the replication and referral-based discovery capabilities to support message integrity and pedigree, and c) implementing anonymization and providing enhanced management and generation of variant configurations.

ABOUT THE AUTHORS

Michael Atighetchi is a scientist at BBN's Information and Knowledge Technologies business unit. His research interests include cross domain information sharing, security and survivability architectures, and middleware technologies. Mr. Atighetchi has published more than 35 technical papers in peer-reviewed journals and conferences, and is a senior member of the IEEE. He holds a master's degree in computer science from the University of Massachusetts at Amherst, and a master's degree in IT from the University of Stuttgart, Germany.

Raytheon BBN Technologies
10 Moulton Street
Cambridge, MA 02138
Phone: (617) 873-1679
Fax: (617) 873-4328
E-mail: matighet@bbn.com

Dr. Joseph Loyall is a Principal Scientist at Raytheon BBN Technologies. He has been the Principal Investigator on DARPA and USAF AFRL R&D projects in the areas of information management, distributed middleware, adaptive applications, and quality of service. He is the author of over 75 published papers; was the program committee co-chair for the Distributed Objects and Applications conference (2002, 2005); and has been an invited speaker at several conferences and workshops. He has a Ph.D. from the University of Illinois.

Raytheon BBN Technologies
10 Moulton Street
Cambridge, MA 02138
Phone: (617) 873-4679
Fax: (617) 873-4328
E-mail: jloyall@bbn.com

Jonathan Webb is an engineer in BBN's Information and Knowledge Technologies business unit. Over 20 years at BBN, Mr. Webb has been involved in a wide range of software development projects including simulation of dynamic systems, web-based data management systems, middleware for information management, and cross domain information sharing. Mr. Webb has a master's degree in aeronautics and astronautics from the Massachusetts Institute of Technology.

Raytheon BBN Technologies
10 Moulton Street
Cambridge, MA 02138
Phone: (617) 873-3321
Fax: (617) 873-4328
E-mail: jwebb@bbn.com

Michael J. Mayhew has been with AFRL for the past 13 years, 6 of those years as a Federal Civilian. As the Program Manager of the Cross-Domain Innovation & Science group, Michael leads a team of research engineers in finding and developing new cross domain technologies and maturing those technologies for integration within existing cross domain products. Michael is a frequent presenter at worldwide program and technical conferences each year and is recognized as a subject matter expert in the area of cross domain technology. In addition to his CDIS role, Michael also serves as the Science & Technology Liaison between his group and the DoDIIS Cross Domain Management Office. Michael's long career included Government Lead Engineer on the ISSE Guard 3.5 project and Senior Computer Analyst with Northrop Grumman Government Systems. Michael received an M.S. with Magna Cum Laude in Computer Science from SUNY IT. Michael is certified ACQ Level 1 Certified Program Manager and ACQ Level 3 Certified SPRD&E. Michael is a member of the Cross Domain Solutions Working Group, and the Armed Forces Communications and Electronics Association, Erie Canal Chapter.

AFRL/RIEB
525 Brooks Road
Rome, NY 13441-4505
Group Tel: (315) 330-7380
Tel: (315) 330-2898 DSN: 587-2898
Fax: (315) 330-3913 DSN: 587-3913
E-Mail: michael.mayhew@rl.af.mil

ACKNOWLEDGMENTS

The authors would like to acknowledge the support and collaboration of the U.S. Air Force Research Laboratory (AFRL) Information Directorate. This material is based upon work supported by the AFRL under Contract No. FA8750-09-C-0012.

NOTES

1. A Domain represents one or more computers under the same specific security policy.
2. A Cross Domain Solution is an approved trusted data flow implemented between two or more domains.
3. A DMZ is a physical or logical subnetwork that contains and exposes an organization's external services to a larger untrusted network, e.g., the Internet.
4. The find_TModel UDDI request is used to retrieve summary information about UDDI tModel elements describing a service.

REFERENCES

1. DoD. (2007) Signed DoDI 8510.01 - Department of Defense Information Assurance Certification and Accreditation Process (DIACAP) Instruction. <http://www.dtic.mil/whs/directives/corres/pdf/851001p.pdf>.
2. NIST. (2009, August) Special Publication 800-53. <http://csrc.nist.gov/publications/PubsSPs.html>.
3. Director of National Intelligence (DNI) Directive, Intelligence Community Directive (ICD) 503, 2008.
4. Boyd Fletcher, USJFCOM J9/SPAWAR, "XMPP & Cross Domain Collaborative Information Environment (CDCIE)," in *Overview For Net-Ready Sensors Summer Workshop Collaboration Gateway*.
5. Chris Roberts, Kurt Risser, Boyd Fletcher, "The Design and Implementation of a Guard Installation and Administration Framework," in SE-Linux Symposium, 2007.
6. JFCOM. (2009, Nov.) Cross Domain Collaborative Information Environment. [Online]. <http://www.jfcom.mil/about/fact_cdcie.html>.
7. Apache Software Foundation. (2010, January) Apache2 Homepage. [Online]. <http://ws.apache.org/axis2/>.
8. Apache Software Foundation. (2010, January) Log4j Homepage. [Online]. <http://logging.apache.org/log4j/1.2/index.html>.
9. Alan Vinh and Phil Bonderud. (2008, Dec.) jUDDI. [Online]. <http://ws.apache.org/juddi/>.
10. OASIS UDDI Spec TC. (2004, June) Using WSDL in a UDDI Registry - Version 2.0.2 - Technical Note. [Online]. <http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm>.

Service Incentive:

Towards an SOA-Friendly Acquisition Process

James T. Hennig, U.S. Army RDECOM CERDEC C2D
Arlene F. Minkiewicz, PRICE Systems LLC

Abstract. Service Oriented Architectures (SOA) offers the DoD the promise of cost savings, data sharing, interoperability, and increasingly agile operations. As with all things that progress in society, there are obstacles. One of the challenges faced by the DoD involves molding current acquisition processes and cultures to be SOA friendly. This paper discusses these challenges and presents some thoughts on how they might be addressed.

Introduction

Metcalf's Law tells us that the value of a telecommunications network is proportional to the square of the number of users of the system [1]. Service Oriented Architectures (SOAs) capitalize on this phenomenon. Through a set of standard interfaces, services (i.e., software-based capabilities) are made available to any consumer willing to follow the structural and behavioral rules for consumption. The loose coupling provided by standard interfaces enables this plug-and-play capability. Taking advantage of such a notion promises great gains in efficiency for anyone looking to create interoperable, scalable applications that share information across boundaries.

According to Gartner, SOA will be used in more than 80% of mission-critical operational applications and business processes by the year 2010 [2]. Analysis of the literature indicates that the SOA vision leads to a belief of implementation efficiencies and cost savings of epic proportions. As the U.S. DoD moves forward with its vision of highly distributed net-centric capabilities in current and future DoD programs, it will be difficult to deploy, maintain, and evolve capabilities without the benefit that SOA brings to the table.

SOA offers the DoD the promise of cost savings, data sharing, interoperability and increasingly agile operations. But, as with all things that progress in society, there are obstacles. The DoD depends on outside contractors to develop much of its needed capabilities. These contracts may involve delivering a specific platform, such as a quantity of F-22s or F-35s, or they may require the delivery of a set of capabilities to satisfy one or many missions such as Future Combat Systems or Distributed Common Ground Systems. The contractors who deliver these capabilities are, not surprisingly, doing so for a profit. With this profit as a motivator, contractors will be unlikely to choose reusing a network-available capability when they can be paid to develop the solution themselves. Incentives are needed to make the existing capability a desirable option for the contractor.

In addition to technical challenges associated with deploying solutions that take advantage of service-oriented technology, there are cultural and organizational challenges that the DoD is likely to encounter. Contractors, who are being paid to deliver a solution or a capability to a specific customer, are unlikely to think beyond their contractual obligations. When developing a service, a contractor will be uninspired to think about the bigger picture, especially in situations where there is schedule pressure or cost containment issues (a frequent occurrence with many DoD software projects).

This paper describes Service Oriented Architecture and the potential value this technology could bring to the DoD. It then addresses the cultural and organizational aspects associated with getting quality SOA solutions within a contract development scenario. Finally, some suggestions are presented for establishing incentives to encourage SOA-friendly behavior within such a scenario.

What is a Service Oriented Architecture?

Service orientation is not a new concept. We are all providers and consumers of services. If I want power for my toaster, I put the plug into the wall socket and power flows. I require no knowledge of how the power gets from the wall socket into the toaster or what substation generates the power. As a service consumer, all I need is the correct interface (my plug) to get access to the electricity, and a Service Level Agreement with the service provider, in this case the electric company, which indicates my willingness to pay for the service. And throughout the U.S., anyone with that same interface and an agreement with their local electric company can get access to power in the same way.

In the context of software, a Service Oriented Architecture is a paradigm that offers software service providers the potential to share their software solutions with consumers using the same basic business model that utilities have used successfully for years. Service consumers are then able to reuse capabilities developed by others rather than having to develop that capability themselves. An SOA is an architectural

style that allows for distribution of capabilities that need not all be supplied or owned by the same organization or entity, with the same notion of transparency that utilities offer electric consumers. From the DoD's perspective, SOA offers the opportunity to create solutions that get the right information to the right places at the right time.

The Value of SOA

SOA results in two distinct categories of software: services (for example web services published in a global directory) that are published and made available by service providers, and software that consumes these services to create capabilities. These software services can be further characterized as either infrastructure services required by many software applications (such as security, messaging, and routing), or business services that are specific to business requirements or specific missions. Compare this to more traditional software paradigms where the business or mission-specific capabilities are closely meshed with software that supports the infrastructure of the application. Separating the infrastructure from the business rules makes it possible to respond quickly as business rules or mission requirements change. SOA creates an environment where the business drives IT requirements rather than being constrained by them..

By definition, SOA services are to be reusable. In an organization as large as the DoD, the existence of reusable services creates many opportunities to reduce redundancy and increase efficiency. From a mission effectiveness perspective, there are many areas where SOA could add value. SOA promises to increase interoperability within and among the services through discoverable standardized service contracts. Through reusable data services, information can be shared across the enterprise increasing dissemination and knowledge transfer. Readiness can be improved through efficiencies gained in information access. Additionally, widespread SOA throughout the DoD will increase organizational ability to deal with rapid change.

The SOA Acquisition Challenge

It's not too hard to see that SOA may add value to the DoD but there are certainly some technological challenges that must be overcome. Challenges aren't going to stop smart software professionals from developing and delivering quality software to the DoD. There are, however, some cultural and organizational challenges that may stand in the way of successful transition to SOA.

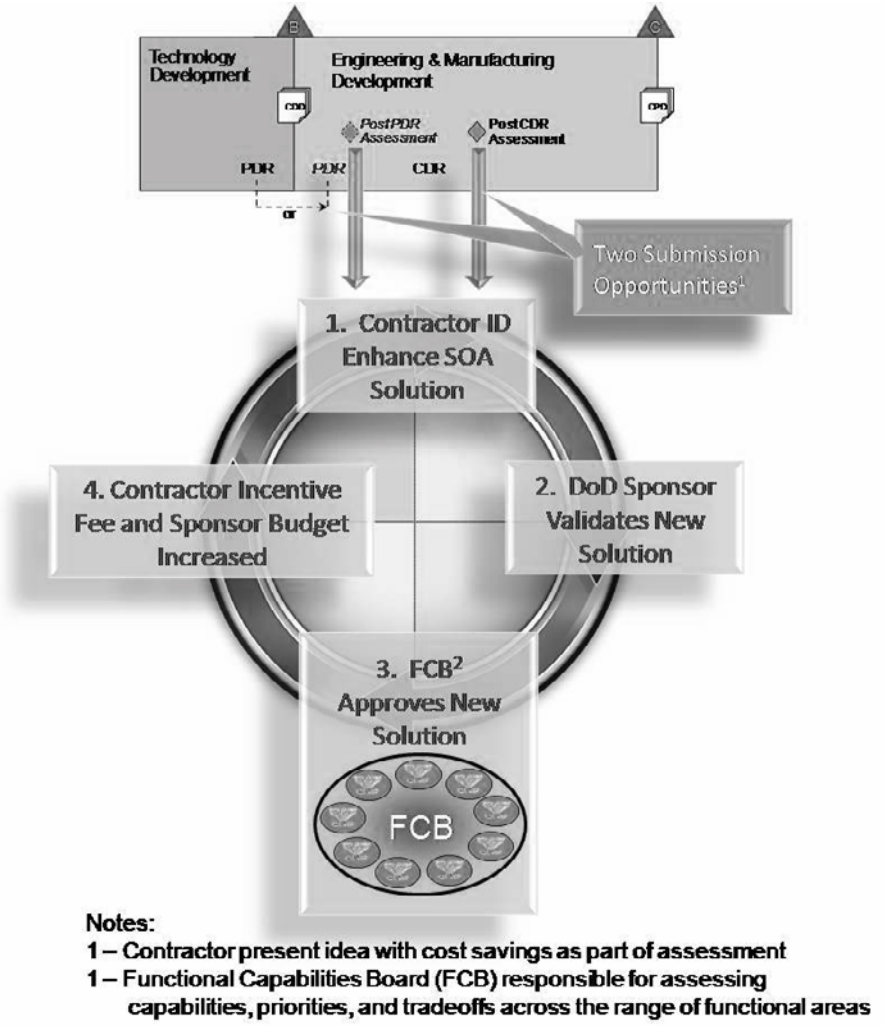
Imagine a contractor who has been awarded the contract (hypothetical) to develop a capability to store food allergy data for all of the Army's soldiers and disseminate this information to all locations where the soldiers are fed—including military bases, theaters of operation, military hospitals, etc. While developing the data services to process this information, the contractor's software engineering team realizes that developing a more generic service to handle all types of allergies—including food, drug, bee stings, etc.—would be a more valuable service to the DoD as a whole. At the same time, the customer program team realizes that this more useful service will take more time and resources to develop; time and resources not currently in the budget. The contractor's customer program team abandons good SOA practices (facilitating a more widely useable service) to create a point solution to the problem because there is no organizational means to quickly adjust the schedule and budget.

This is, of course, a very simplified example—many opportunities will arise that could provide useful solutions throughout the DoD that may be overlooked because funding is targeted at specific capabilities. A project is not service-oriented just because capabilities are delivered using sharable services. A project is not truly service-oriented unless it takes advantage of existing services where available and develops needed services taking into account the bigger picture of uses beyond the current need. DoD contracts focus on the particular capability being contracted for and make no provisions for delivering beyond that. Contractors are paid for the capability they deliver, making it desirable to maximize capability developed for a specific contract. This is not to suggest that the contractors for particular projects should be responsible for the creation and maintenance of an SOA framework suitable to meet DoD requirements. Contractors working on specific projects should intend to take advantage of existing DoD SOA frameworks. Contractors however, should be encouraged to embrace SOA for their projects by leveraging the use of services existing within that framework and considering the greater good when developing new services to be made available through that framework.

In this way, SOA creates a paradox for the DoD and its contractors. The DoD has specific capabilities that it knows it needs and it has a time frame and budget within which it expects to meet those needs. Within the DoD, the "sponsor" of a specific capability will outsource the fulfillment of this capability to a community of engineers, designers and other software development personnel. Neither the sponsor nor the contractor is rewarded or incentivized to provide a service-based solution, which meets a greater good and provides additional enterprise benefit for the whole of the DoD. There are limited explicit incentives to take advantage of existing services when possible that meet program needs. The DoD has unwittingly tied the hands of these very talented professionals by not providing a mechanism to encourage a specific focus on enterprise benefit.

Cultural and organizational changes are necessary if the DoD is going to be successful with full-scale SOA solutions. Contractors and project sponsors should be encouraged through policy changes and funding incentives to think beyond the current problem. Both the contractor and customer sponsor need to be incentivized to develop services that will solve problems the DoD might not yet realize that they have—

Figure 1



or issues that might not be relevant to the contracting agency but that could have significant impact on another agency. Suppose there was a process through which contractors can come back to the table during the planning and requirements phases of a project with suggestions for a better, more far-reaching SOA solution than that which was originally contracted. Figure 1 depicts a notional process.

Contractors should be given opportunities to identify enhanced SOA solutions to the contracting agency. This opportunity could be presented to the DoD sponsors, outlining additional costs as well as added value of the enhanced solution. Additionally the contractor should present the cost savings anticipated if the enhanced service is provided in the context of the current program versus having to do it separately as a new program or upgrade. Once the DoD sponsor validates the new solution, the improvements would be passed on to the Functional Capabilities Board for approval. Ideally, the contractor and the DoD sponsor would be given the opportunity through this mechanism to present suggestions not only to the contracting agency, but to other branches of the DoD that might benefit from such a service. Upon validation of the value added by the new service, a portion of the cost savings incurred could then be provided as both an award fee incentive to the contractor and a budget increase to the sponsor.

There should also be incentives for contractors to include reuse of existing services as part of their bid for the contract. Contractors should be encouraged to work with the contracting agencies and a Functional Capabilities Board to identify services existing in either the DoD or the public domain that would be suitable in the context of the current contract. Contract awards should include provisions for a "finder's fee" based on the anticipated savings to the contracting agency, taking into consideration not only reduced costs for the current program but also recognizing the value in non-duplication of services.

Conclusion

SOA is likely here to stay. It offers great opportunities for the Services and the entire DoD to develop forward-thinking synergistic solutions that transcend current operational requirements. In order for this to happen, the DoD needs to find ways to encourage contractors and DoD sponsors to embrace SOA beyond just the "letter of the law" to the point where they are architecting solutions designed to take advantage of the benefits and cost savings possible with SOA. On the other hand, contractors need to be proactive in their approach to providing quality SOA solutions to the DoD that consider requirements beyond a current contract and look to how contract solutions can add value beyond that contract to other applications across the DoD enterprise.

As SOA evolves within the DoD, acquisition culture needs to shift to enable collaborative behavior that will provide solution synergy. The DoD will benefit by getting the most value out of services contracted for particular programs. The contractors benefit as their proactive behavior in defining opportunities makes them a vital part of the DoD's SOA planning process, bringing them to the table as the DoD works to create SOA Advisory Boards and SOA Centers of Excellence.

ABOUT THE AUTHORS

James T. Hennig is the Chief Architect for the Battle Command Division of the U.S. Army Research Development and Engineering Command, Communication, Electronics, Research, Development and Engineering Center, Command and Control Directorate. He has a B.S. in Mechanical Engineering, an M.S. in Software Engineering and is currently working on a Ph.D. in Systems Engineering and Enterprise Systems. He has 20 years experience building highly complex distributed computing systems.

732-427-3088
James.Hennig@us.Army.Mil

Arlene Minkiewicz is the Chief Scientist at PRICE Systems, LLC. In this role, she leads the cost research activity for the entire suite of cost estimating products that PRICE provides. Ms. Minkiewicz has more than 25 years of experience with PRICE building cost models. She has a B.S. in Electrical Engineering and an M.S. in Computer Science. Minkiewicz has published many articles on software measurement and estimation and frequently presents her research at industry forums.

856-608-7222 (work)
856-630-9408 (cell)
17000 Commerce Parkway, Suite A
Mt. Laurel, NJ 08054
arlene.minkiewicz@pricesystems.com

REFERENCES

1. Wenzel, G. & Yuan, E. Actionable Intelligence for the Warfighter – Achieving Army ISR Net-Centricity Through a Service Oriented Architecture available at <http://asc.army.mil/docs/pubs/alt/current/issue/articles/14_Actionable_Intelligence_for_the_Warfighter_--_Achieving_Army_ISR_Net-Centricity_Through_a_Service-Oriented_Architecture_(SOA)_200704.pdf>.
2. Gartner report available at <http://www.gartner.com/it/page.jsp?id=503864>.

FURTHER READING

1. McGovern, J, et. al., *Java Based Web Applications*, Elsevier Science, 2003.
2. Web Services and Service-Oriented Architectures <http://www.service-architecture.com>.
3. Chopra, D., "Security for SOA and Web Services", Dec 2004, available at <https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/512de490-0201-0010-ffb4-8bd1620b2386>.
4. NCES, "NCES Security Server", Defense Online, July 2006, available at <http://ges.dod.mil/ServiceSecurity.htm>.
5. Lewis, G., et.al., "SMART : The Service-Oriented Migration and Reuse Technique", September 2005, available from the SEI at <http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn029.pdf>.
6. Starret, E., "Software Acquisition in the Army", *CrossTalk*, May 2007.
7. Zenishek, Lt. Col S., Usechak, Dr. D., "Net-Centric Warfare and its Impact on System-Of-Systems", *Defense Acquisition Review Journal*, 2005, available at <http://www.dau.mil/pubs/arq/2005arq/2005arq-39/Zenishek.pdf>.

Global Workforce Development Projects in Software Engineering

Art Pyster, Stevens Institute of Technology
Mark Ardis, Stevens Institute of Technology
Dennis Frailey, Raytheon and Southern Methodist University
David Olwell, Naval Postgraduate School
Alice Squires, Stevens Institute of Technology

Two projects now underway have the potential to significantly improve the worldwide software engineering workforce. The Integrated Software and Systems Engineering Curriculum Project (ISSEC) recently published *Graduate Software Engineering 2009 (GSwE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering*. Initially sponsored by DoD with over 40 authors, the IEEE Computer Society and the Association for Computing Machinery now maintain and evolve GSwE2009 with support from the International Council on Systems Engineering (INCOSE). The second project, Body of Knowledge and Curriculum to Advance Systems Engineering (BKCASE), is creating two products: a body of knowledge for systems engineering and guidelines for a professional master's degree in systems engineering. Both the body of knowledge and the reference curriculum will incorporate software engineering as appropriate, to reflect the critical importance that software plays in modern systems. DoD, INCOSE, IEEE Systems Council, and IEEE Computer Society Educational Activities Board support and participate in BKCASE. Together, the products of ISSEC and BKCASE should accelerate the collaboration and potential integration of the systems and software engineering workforces.

Improvements in the software engineering workforce that support the DoD and its contractor community depend, in part, on the strength of community agreements on how to educate, guide, inform, evaluate, and certify the workforce. Two projects with broad community involvement are providing some of those agreements:

1) **ISSEC**
2) **BKCASE**

ISSEC Summary

ISSEC was launched by Art Pyster at the Stevens Institute of Technology (Stevens) in 2007 with DoD sponsorship and a coalition from academia, industry, government and professional societies providing authors. In September 2009, its more than 40 authors published version 1.0 of a reference curriculum that reflects current development practices and the greater role of software in today's systems. The report, titled *Graduate Software Engineering 2009 (GSwE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering* [1], is available at <http://www.gswe2009.org>. Two companion documents followed in November 2009, Comparisons of GSwE2009 to Current Master's Programs in Software Engineering and Frequently Asked Questions on Implementing GSwE2009. Both are also available on the GSwE2009 website.

ISSEC continues today, focused on aiding dissemination and adoption of GSwE2009.

The IEEE Computer Society and the Association for Computing Machinery (ACM) have recently signed a copyright transfer agreement with Stevens to become the owners and primary sponsors of GSwE2009. The two professional societies now assume responsibility for evolving and maintaining the guidelines to the same level that they manage curriculum guidelines in other disciplines. The International Council on Systems Engineering is playing a supporting role in the evolution of GSwE2009. Stevens and a number of the original author team members maintain purview over the two companion documents.

BKCASE Summary

BKCASE began in September 2009 under the joint leadership of Art Pyster from Stevens and Dave Olwell from the Naval Postgraduate School. As did ISSEC earlier, BKCASE has enjoyed strong support from both DoD and INCOSE since the project began. The IEEE Systems Council and the IEEE Computer Society Educational Activities Board offered their support for BKCASE in November 2009. As of the writing of this paper, BKCASE has 45 authors from 10 countries, and is supported by over a hundred reviewers.

BKCASE will produce two primary products:

1) **Systems Engineering Body of Knowledge (SEBoK— pronounced "sea" "Bach")**
2) **Graduate Reference Curriculum for Systems Engineering (GRCSE— pronounced "Gracie")**

In the second half of 2010, BKCASE will publish version 0.25 of both the SEBoK and GRCSE. Version 1.0 will follow sometime in 2012. BKCASE will, quite naturally, turn to SEBoK for the material that should be included in GRCSE. Both products will incorporate substantial aspects of software engineering, which will help bridge the historical gap between professional software and systems engineers.

The ISSEC Project

In 1989 the SEI of Carnegie Mellon University published a landmark report on graduate education in software engineering [2]. Several universities used the recommendations in that report to establish their software-engineering degree programs. Since then, the way software is developed has changed dramatically, yet little effort has been made to foster further implementation and update the Software Engineering Institute's (SEI) original recommendations for graduate education in software engineering [2].

In 2007, Kristen Baldwin, then Deputy Director for Software Engineering and System Assurance of the Office of the Under Secretary of Defense Acquisition, Technology and Logistics, approached Art Pyster of Stevens Institute regarding the findings of a software industrial base study that had been conducted at the request of the Office of the Secretary of Defense. The study reflected that software drives the performance of almost all major military systems today and the development phase of any major system typically involves substantial amounts of software development. The study found a critical shortage of trained senior-level software talent required by the complex, software-intensive systems developed and forecasted by the Department of Defense.

Baldwin and Pyster concluded that a critical long-term strategy for the DoD was to ensure a strong and relevant foundation for training and education of senior software talent through establishment of a reference curriculum that would represent the fundamentals of software engineering as well as address the current challenges of scale, complexity, and criticality. Based on these conclusions, ISSEC began.

ISSEC built GSwE2009 on the SEI curriculum plus those of other initiatives, such as the Guide to the Software Engineering Body of Knowledge (SWEBOK) [3] and Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering [4]. ISSEC followed an iterative, evolutionary approach in creating the guidelines, beginning with the formation of a Curriculum Author Team (CAT). First established in July 2007, the CAT is a collection of invited experts from industry, government, academia, and professional associations. CAT membership grew as GSwE2009 matured. In addition to representatives from the ACM, IEEE Computer Society, and INCOSE, ISSEC had the benefit of authors from the Brazilian Computer Society and the U.S. National Defense Industrial Association Systems Engineering Division.

Originally, GSwE2009 was known as GSwERC, which stands for Graduate Software Engineering Reference Curriculum. The CAT released GSwERC 0.25 in February 2008, GSwERC 0.5 in October 2008, and GSwE2009 1.0 in September 2009. The software engineering community was invited to review versions 0.25 and 0.5 to provide the necessary feedback to develop version 1.0. The review of version 0.5 generated more than 800 individual review comments, which were adjudicated for use in creating version 1.0. The detailed comments and their adjudication can be found on the GSwE2009 website.

GSwE2009 Content

GSwE2009 includes the following elements:

- **>> A set of outcomes to be fulfilled by a student who successfully completes a graduate program based on the curriculum**
- **>> A set of student skills, knowledge, and experience assumed by the curriculum, not intended as entrance requirements for a specific program, but as the starting point for the curriculum's outcomes**
- **>> An architectural framework to support implementation of the curriculum**
- **>> A description of the fundamental or core skills, knowledge, and practice to be taught in the curriculum to achieve the outcomes. This is termed a Core Body of Knowledge (CBOK) and includes topic areas and the depth of understanding a student should achieve**

A university considering the creation or modification of a graduate software engineering program should be able to use the CBOK and the architectural framework to design appropriate courses and degree requirements. The outcomes and entrance assumptions should help in determining the expected market and value of the program to potential students and their employers.

In addition, GSwE2009 includes the following:

- **>> The fundamental philosophy for GSwE2009 development as described in a set of guiding principles**
- **>> A discussion of how GSwE2009 will evolve to remain effective**
- **>> A mapping of expected outcomes to the CBOK and to the total GSwE2009 program recommendations**
- **>> A description of Knowledge Areas discussed in GSwE2009 that are not yet fully integrated into the current version of the SWEBOK**
- **>> Glossary, references, and other supporting material**

Expected Student Outcomes

Graduates of a master's program that satisfies GSwE2009 will do the following:

- **>> Master the CBOK**
- **>> Master software engineering in at least one application domain, such as finance, medical, transportation, or telecommunications; and one application type, such as real-time, embedded, safety-critical, or highly distributed systems. That mastery includes understanding how differences in domain and type manifest themselves in both the software and the engineering of the software, and includes understanding how to learn a new application domain or type**

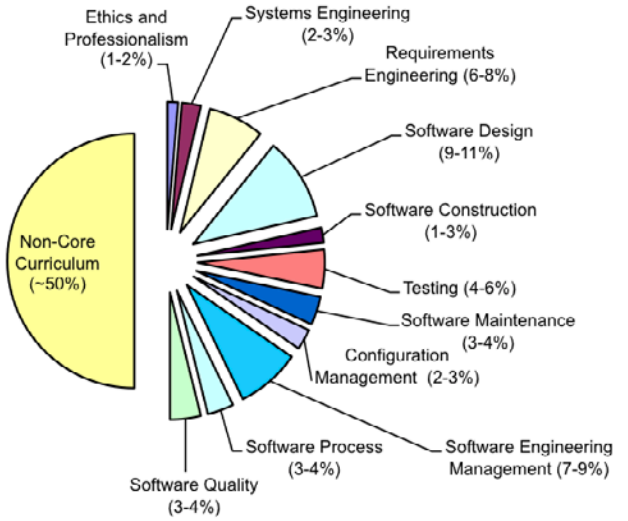
- **Master at least one Knowledge Area or sub-area from the CBOK to at least the Bloom Synthesis level [5]**
- **Be able to make ethical professional decisions and practice ethical professional behavior**
- **Understand the relationship between software engineering and systems engineering and be able to apply systems engineering principles and practices in the engineering of software**
- **Be an effective member of a team, including teams that are international and geographically distributed; effectively communicate both orally and in writing; and lead in one area of project development, such as project management, requirements analysis, architecture, construction, or quality assurance**
- **Be able to reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems, and organizations**
- **Understand and appreciate feasibility analysis, negotiation, and good communications with stakeholders in a typical software development environment, and be able to perform those tasks well; have effective work habits; and be a leader**
- **Be able to learn new models, techniques, and technologies as they emerge, and appreciate the necessity of such continuing professional development**
- **Be able to analyze a current significant software technology, articulate its strengths and weaknesses, compare it to alternative technologies, and specify and promote improvements or extensions to that technology**

Core Body of Knowledge

The CBOK includes all of the fundamental or core skills, knowledge, and experience to be taught in the curriculum to achieve the expected student outcomes. The primary source for developing the CBOK was the SWEBOK. Knowledge elements were also derived from the Software Engineering 2004 curriculum guidelines [4], the INCOSE Guide to Systems Engineering Body of Knowledge [6] and especially the INCOSE Systems Engineering Handbook [7].

Figure 1 shows the knowledge elements of CBOK and their expected relative proportions of the GSwE2009 curriculum. Although specific systems engineering knowledge elements only represent 2–3% of the CBOK, they are considered a cross-cutting concern that arises in many other areas. For example, systems engineering material would also be covered under requirements engineering, testing, configuration management and project management.

Figure 1. CBOK knowledge elements as percentages of GSwE2009 curriculum

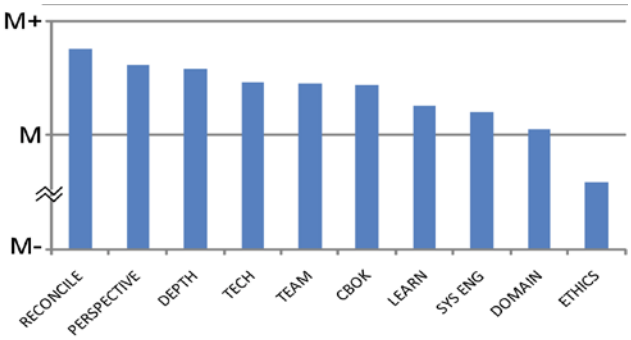


Companion Reports

In addition to GSwE2009, ISSEC has published two companion reports on its website: Comparisons of GSwE2009 to Current Master's Programs in Software Engineering and Frequently Asked Questions on Implementing GSwE2009. The latter report is intended to help schools establish or modify a graduate software engineering program to align with the new curriculum recommendations.

The comparison report provides information on about a dozen current programs. Since most programs have alternative tracks, two or three hypothetical students from each of these schools are described. Using the courses in their individual programs, an assessment is made of each student's ability to achieve the new recommended outcomes. While all programs compare fairly well, all had areas where they could improve. For example, most programs do not cover ethics or systems engineering topics as thoroughly as recommended by GSwE2009.

Figure 2. Average Outcome Fulfillment



Comparison of GSwE2009 Guidance and Actual Programs

GSwE2009 comparisons were performed in collaboration with representatives of 12 currently offered software engineering programs, nine from North America. The focus was on comparison of the 10 GSwE2009 outcomes with the expected outcomes currently attained by up to three diverse, hypothetical, but typical, students from each program. Many interesting facts were learned about the differences among current software engineering programs, but space does not permit further elaboration here.

By the GSwE2009 guidelines, the programs examined clearly do a reasonable job of satisfying the outcomes to a "medium" level, at least for the "typical" students described. As shown in Figure 2, each program had some room for improvement to fully meet all GSwE2009 outcomes for most students. The outcomes least likely to be attained at a higher level are ethics (few programs offer much coverage of this), systems engineering (many programs cover this topic only lightly) and application domain depth (some of the programs do not afford their students an opportunity to attain such depth).

Similar entrance requirements (required degrees, levels of experience, etc.) do not always correspond to similar levels of outcome attainment, even when the students appear to have similar backgrounds. Individual programs vary greatly from one another in the overall outcome attainment levels of their students, but most programs do make a difference—that is, outcome attainment upon graduation is typically much higher than upon entry. Industry experience typically results in higher outcome attainment. Hypothetical students within most of the programs vary in their levels of outcome attainment, suggesting that their choices of electives and tracks make a significant difference.

The most commonly required courses are software project management, software architecture and design, software requirements, and testing or verification and validation. By contrast, relatively few programs require courses in construction, metrics, ethics, or systems engineering.

Data from three non-U.S. programs suggest that there are significant differences of perspective, and that the GSwE2009 model is more U.S.-centric than originally intended.

The BKCASE Project

BKCASE, which began in September 2009, will generate two related products by 2012—SEBoK and GRCSE. BKCASE is organized along similar lines to ISSEC. A diverse author team, currently composed of 45 people from 10 countries, meets face to face every three months and works in smaller groups via collaboration technology between workshops. The first author workshop was held at the Naval Postgraduate School in December 2009, refining and ratifying the project charter, project scope, and resulting in the formation of early teams to begin writing the SEBoK. Teams began working on GRCSE at the second workshop, held at the end of March 2010 at Embry-Riddle Aeronautical University.

As with ISSEC, BKCASE products will initially be owned and managed by the author team and copyrighted by Stevens. Ultimately, SEBoK and GRCSE will have the greatest impact if major professional societies become their "stewards," responsible for their evolution and maintenance. INCOSE and the IEEE Systems Council or Computer Society are the most natural stewards. These societies have several authors participating in BKCASE. See <http://www.bkcase.org> for more complete and current information.

Systems Engineering Body of Knowledge

Readers will benefit from a body of knowledge in systems engineering as described in the value proposition for SEBoK:

>> **There is no authoritative source that defines and organizes the knowledge of the systems engineering (SE) discipline, including its methods, processes, practices, and tools. The resulting knowledge gap creates unnecessary inconsistency and confusion in understanding the role of SE in projects and programs; and in defining SE products and processes. SEBoK will fill that gap, becoming the "go to" SE reference.**

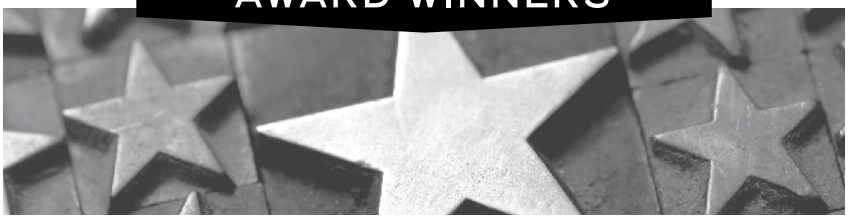
>> **The process of creating the SEBoK will help to build community consensus on the boundaries and context of SE thinking. It will also help the community understand and improve the ability of management, science and engineering disciplines to work together.**

>> **Having a common way to refer to SE knowledge will facilitate communication among systems engineers and provide a baseline for competency models, certification programs, educational programs, and other workforce development initiatives around the world. Having common ways to identify metadata about SE knowledge will facilitate search and other automated actions on SE knowledge.**

At the first author workshop, the authors confirmed this value proposition and that there are two disciplines related to SE that require special attention in the SEBoK—software engineering and project management. Software engineering was singled out because the functionality and character of virtually every interesting system these days relies on software. Software drives much of the architecture, security, safety, scalability, interface, and countless other characteristics of modern systems. Much, if not the majority of the risk and cost of systems development rests with the software elements. Given the enormous impact of software on systems, the SEBoK will contain, in integral fashion, software engineering knowledge. At the first workshop, however, no decisions were made on how to accomplish the integration of software engineering knowledge or project management into the SEBoK.

For Version 0.25, the SEBoK will be domain independent. There will be no effort to define knowledge areas in terms or methods that are specific to a particular domain such as finance, medical devices or defense systems. Domain-specific knowledge will be discussed in companion case studies,

AWARD WINNERS



DoD Systems Engineering Top 5 Program Awards

Sponsored by Department of Defense Systems Engineering Directorate and National Defense Industrial Association Systems Engineering Division

The awards, presented to both government and industry, recognize significant systems engineering achievement by teams of industry and government personnel.

Winners:

- Army: Advanced Field Artillery Tactical Data System (AFATDS)
- USAF: Battlefield Airborne Communications Node (BACN) JUON
- Army: Base Expeditionary Target & Surveillance Systems-Combined (BETSS-C)
- Army: Defense Readiness Reporting System-Army (DRRS-A)
- USAF: C-17 Globemaster III Modernization

Awards presented at the annual NDIA Systems Engineering Conference
San Diego, CA, October 25–28, 2010

http://www.acq.osd.mil/se/apr/top5awards.html

which will address a few domains and walk through how their methods, processes, and terminology align with SEBoK. This decision will be revisited after the release and review of Version 0.25.

Graduate Reference Curriculum for Systems Engineering

Readers will benefit from a graduate reference curriculum in systems engineering as described in the value proposition for GRCSE:

>> **There is no authoritative source to guide universities in establishing the outcomes graduating students should achieve with a master's degree in SE, nor a guidance source on reasonable entrance expectations, curriculum architecture, or curriculum content**

>> **This gap in guidance creates unnecessary inconsistency in student proficiency at graduation; makes it harder for students to select where to attend; and makes it harder for employers to evaluate prospective new graduates**

>> **GRCSE will fill that gap, becoming the "go to" reference to develop, modify, and evaluate graduate programs in SE.**

GRCSE will be based on the SEBoK and will be analogous to GSwE2009 in form. It will define the entrance expectations, curriculum architecture, curriculum content, and expected student outcomes for graduate programs in SE. GRCSE will recommend that students know or learn about the application of SE in an application domain or business segment. The use of GRCSE for guidance will enable consistency in student proficiency at graduation, making it easier for students to select where to attend and for employers to evaluate prospective new graduates. Naturally, based on the earlier comments about the ties between software and systems engineering, GRCSE will weave education on software engineering into its recommendations for graduate students studying systems engineering.

Summary

The development of a high-performance systems and software engineering workforce in a world of increasing complexity requires a foundation of authoritative knowledge and guidance in systems and software. Nowhere is this more vital than with the U.S. military, which develops many of the largest and most complex systems in the world. Two projects, the ISSEC and the BKCASE have stepped up to the challenge of building this foundation. ISSEC published GSwE2009: Curriculum Guidelines for Graduate Degree Programs in Software Engineering to provide authoritative guidelines—based on the current and impacting the future revision of the software engineering body of knowledge—on the development of graduate software engineering curriculum. BKCASE will produce both a SEBoK and a GRCSE by 2012. Together, these projects and products support the development of a strong global software engineering workforce and a systems engineering workforce with the necessary software engineering skills to solve tomorrow's global systems problems.

In addition, readers are encouraged to consider some of the following ways to use the guidelines produced by these projects:

>> **To use as a reference for locating technical information about systems engineering**

>> **To inform their workforce of development efforts**

>> **To assess the educational background of their technical staff**

>> **To develop continuing education curricula or courses for their technical staff**

>> **To advise local universities or training vendors regarding the kinds of courses and/or educational programs needed by their technical staff and future hires, and to use as a framework for selecting educational programs for employees**

>> **To define qualifications for contracted workforce**

Any reader who is interested in contributing to either project or adopting any of the resulting products should send an e-mail with background information and areas of interest to bkcase@stevens.edu.

ABOUT THE AUTHORS

Mark Ardis is a Distinguished Service Professor at Stevens Institute of Technology. He has helped create academic programs in software engineering at Wang Institute of Graduate Studies, Carnegie Mellon University, Rose-Hulman Institute of Technology, Rochester Institute of Technology and Stevens Institute of Technology.

Dennis Frailey is a Principal Fellow at Raytheon and an Adjunct Professor of Computer Science and Engineering at Southern Methodist University. Dennis was lead author of the GSwE2009 Comparisons volume. Dennis is a Fellow and former Vice President of ACM, IEEE Senior Member, and is active in several professional committees, including the IEEE Computer Society Educational Activities Board and the curriculum author team for GSwE2009. He holds an MS and PhD in computer science (Purdue) and a BS in mathematics (Notre Dame).

David. H. Olwell is Professor of Systems Engineering at the Naval Postgraduate School, where he recently completed a five-year term as department chair. His research interests are reliability engineering and statistical quality control. He previously was on the faculty of the United States Military Academy.

Art Pyster is a Distinguished Research Professor at Stevens Institute of Technology and Deputy Executive Director of the Systems Engineering Research Center. Previously he served in a number of senior technical and executive roles including Deputy Chief Information Officer for the U.S. Federal Aviation Administration and Senior Vice President and Director of Systems Engineering and Integration at SAIC.

Alice Squires is a PhD candidate and faculty member in Systems Engineering at Stevens Institute of Technology. She has over 28 years of experience and has served as a technical lead for IBM, a Senior Systems Engineering manager for both Lockheed Martin and General Dynamics, and a Senior Systems Engineer consultant to Lockheed Martin, IBM, and EDO Ceramics.

REFERENCES

1. Pyster, A. (Ed.), *Graduate Software Engineering 2009 (GSwE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering*, Integrated Software & Systems Engineering Curriculum Project, Stevens Institute of Technology, September 30, 2009. <http://www.gswe2009.org>.
2. Ardis, M., and G. Ford. 1989. *SEI report on graduate software engineering education.* CMU/SEI 89-TR-21. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.
3. Bourque, P., and R. Dupuis, eds. 2004. *SWEBOK: Guide to the Software Engineering Body of Knowledge.* Los Alamitos, CA: IEEE Computer Society Press.
4. ACM and IEEE (ACM/IEEE Computer Society Joint Task Force on Computing Curricula). 2004. *Software engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering.* <http://www.acm.org/education/curricula-recommendations>.
5. Bloom, B. S., ed. 1956. *Taxonomy of educational objectives: The classification of educational goals; Handbook I, cognitive domain.* Place: Longmans.
6. INCOSE. 2004. *Guide to Systems Engineering Body of Knowledge.*
7. Haskins, C., ed. 2007. *Systems engineering handbook: A guide for system life cycle processes and activities.* Version 3.1. Rev. by K. Forsberg and M. Krueger. Seattle: INCOSE.

Standing On the Shoulders of Giants!

Since 1997, I have been writing BackTalk columns on a semi-regular basis. I tend to work best under pressure (translation–I have raised procrastination to an art form) so I typically start writing the column about a day before it is due. There is nothing like sheer stress and a looming deadline to sharpen up my writing skill. But I start thinking about the column weeks in advance–writing in my head, discarding ideas. This column, I was stumped. I could not come up with just the right idea. As I was teaching Software Engineering today, I was going over some pithy and somewhat painfully funny quotes covering programming and software engineering. I realized that the column was writing itself–and others were writing it for me! After a little research looking for quotes that sum up our profession, here is my contribution.

Requirements

"If you don't know where you're going, you're unlikely to end up there." - Forrest Gump

"All projects are iterative–it's just that some managers choose to have the iterations after final delivery." - Urban Wisdom

"It is easier to change the specification to fit the program than vice versa." - Author Unknown

"When somebody begins a sentence with 'It would be nice if...' the right thing to do is to wait politely for the speaker to finish. No project ever gets around to the it-would-be-nice features: or it they do, they regret it. Wait for sentences that begin 'We have to...' and pay close attention, and see if you agree." - Tom Van Vleck

Cost Estimation

"It always takes longer than you expect, even when you take Hofstadter's Law into account." - Hofstadter's Law, Douglas Hofstadter

Design

"Programs must be written for people to read, and only incidentally for machines to execute." - Abelson and Sussman

"The hardest part of design … is keeping features out." - Donald Norman

"A designer can mull over complicated designs for months. Then suddenly the simple, elegant, beautiful solution occurs to him. When it happens to you, it feels as if God is talking! And maybe He is." - Leo Frankowski (in The Cross-Time Engineer)

"The purpose of analysis is not modeling but understanding." - Sun Tsu, The Art of War

Programming and Programming Languages

"C++ would make a decent teaching language if we could teach the ++ part without the C part." - Michael B. Feldman

"It has been discovered that C++ provides a remarkable facility for concealing the trivial details of a program–such as where its bugs are." - David Keppel

"And then it occurred to me that a computer is a stupid machine with the ability to do incredibly smart things, while computer programmers are smart people with the ability to do incredibly stupid things. They are, in short, a perfect match." - Bill Bryson

"Good code is its own best documentation. As you're about to add a comment, ask yourself, 'How can I improve the code so that this comment isn't needed?'" - Steve McConnell

"The only way for errors to occur in a program is by being put there by the author. No other mechanisms are known. Programs can't acquire bugs by sitting around with other buggy programs." - Harlan Mills

"There are two ways to write error-free programs; only the third one works." - Alan J. Perlis

"There does not now, nor will there ever exist, a programming language in which it is the least bit hard to write bad programs." - Lawrence Flon

"That's the thing about people who think they hate computers. What they really hate are lousy programmers." - Larry Niven

"The evolution of languages: FORTRAN is a non-typed language. C is a weakly typed language. Ada is a strongly typed language. C++ is a strongly hyped language." - Ron Sercely

"You can tell how far we have to go, when FORTRAN is the language of supercomputers." - Steven Feiner

Reuse

"I've finally learned what 'upward compatible' means. It means we get to keep all our old mistakes." - Dennie van Tassel

Testing

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." - Brian W. Kernighan.

"Software undergoes beta testing shortly before it's released. Beta is Latin for 'still doesn't work.'" - Author Unknown

"If debugging is the process of removing bugs, then programming must be the process of putting them in." - Author Unknown

"It's not a bug, it's an undocumented feature." - Author Unknown

Maintenance

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live." - Martin Golding

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to produce bigger and better idiots. So far, the universe is winning." - Author Unknown

Software Engineering in General

"If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization." - Gerald Weinberg

"It has been said that the great scientific disciplines are examples of giants standing on the shoulders of other giants. It has also been said that the software industry is an example of midgets standing on the toes of other midgets." - Alan Cooper

If you have other "Great Quotes," please send them to me and I'll publish "Quotes Volume II" in a later column!

David A. Cook
Stephen F. Austin State University
cookda@sfasu.edu

23rd Annual

SSTC
Systems & Software Technology Conference

Plan now to join us for excellent, quality presentations and networking with colleagues from military/government, industry and academia.

Topics Include…

* Concepts and Trends
  - Example Areas:
    - Cloud Computing
    - Model Driven Processes
    - Multi Processor Challenges
    - SOA
* Cyber Technologies
  - Example Areas:
    - Cyber Security
    - Cyber Defense
    - Cyber Physical Systems
* Guidance, Policies, and Standards
* Human Capital / Workforce Development
* Modernization of Systems
* Real World Lessons
* Research
* Social Networking
* Technological Tool Advances
  - Example Areas:
    - Acquisition Processes
    - Agile Development
    - Assessments
    - Data Development & Environments
    - Program Management & Methods
    - Rugged and Resilient Systems
    - Smart Grids
    - Systems & Systems Assurance
    - Software
    - Testing Methods
    - Verification / Validation
    - Web Authentication

Conference Registration Opens 24 January 2011

Mark your calendar!

For conference & trade show information, visit
WWW.SSTC-ONLINE.ORG

Ok, we've all seen it, individuals totally immersed in hand-held devices consuming an endless stream of information and knowledge - thumbs furiously dancing across miniscule keyboards entering thoughts, ideas, and directions.

Much like the tiny information exchange tools meant to keep us up-to-speed in this data-filled world, SSTC 2011 will focus on connecting attendees with technological advancements associated with building better systems and software in support of our defense forces.

Join us 16 – 19 May 2011
in Salt Lake City, Utah, as we Sync-Up with advances in Technology!

CrossTalk / 517 SMXS MXDEA
6022 Fir Ave.
BLDG 1238
Hill AFB, UT 84056-5820

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

CrossTalk thanks the above organizations for providing their support.