CrossTalk

March / April 2017 The Journal of Defense Software Engineering Vol. 30 No. 2

Process Improvement Best Practices:

Modern Process Trends

TABLE OF CONTENTS

Departments

Cover Design by Kent Bingham

Modern Process Trends

CrossTalk

NAVAIR Jeff Schwalb
309 SMXG Kelly Capener
76 SMXG Mike Jennings

Publisher Justin T. Hill
Article Coordinator Heather Giacalone
Managing Director David Erickson
Technical Program Lead Thayne M. Hill
Managing Editor Mary Harper
Copy Editor Breanna Olaveson
Senior Art Director Kevin Kiernan
Art Director Mary Harper

Phone 801-777-9828
E-mail Crosstalk.Articles@hill.af.mil
CrossTalk Online www.crosstalkonline.org

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); and U.S. Air Force (USAF). USN co-sponsor: Naval Air Systems Command. USAF co-sponsors: Ogden-ALC 309 SMXG and Tinker-ALC 76 SMXG.

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: **CrossTalk** 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:
For questions or concerns about crosstalkonline.org web content or functionality contact the **CrossTalk** webmaster at 801-417-3000 or webmaster@luminpublishing.com.

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

CrossTalk is published six times a year by the U.S. Air Force STSC in concert with Lumin Publishing <luminpublishing.com>. ISSN 2160-1577 (print); ISSN 2160-1593 (online)

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

Since the early days of the quality revolution, improvements and efficiencies have been in the forefront of developing engineering disciplines. Various standards, guidelines, and models have employed learned best practices from government and industry to continually evolve and change our methodologies, technologies and lifecycles. Exciting new trends are emerging in architectural solutions, lean concepts, agile development, business process models, etc.

In 1989, when Watts Humphrey published his book "Managing the Software Process," it was widely perceived that software was considered more of an arcane art than an engineering discipline. He suggested that software should be managed like other engineering efforts and suggested a five-step maturation scale for software organizations. Fast forward to today's set of practices where changes in processes and technologies have yielded the ability to manage software from concept to delivery in weeks utilizing agile methods that have moved from manifesto to mainstream. Our understanding of how processes are defined and utilized maximize our ability to produce high quality software in a fast-paced, ever-changing world.

In this issue of Crosstalk, we explore some of the technologies and techniques that continue to improve our capabilities. One great example is Wayne Abba's examination of traditional earned value methods and how the concepts can still be utilized to great benefit, even in a more agile setting. Changes to the software lifecycle are imperative to support current development needs which Don O'Neill details in his article "In Search of a Modern Software Lifecycle." Dr. Nary Subramanian's treatise on improving and even automating of the software deployment process examines software improvements prior to delivery to the end user. Michael Elliott discusses the employment of modern software engineering techniques to critical systems, such as civil airborne systems. And the processes of process improvement are changing, as evidenced by Richard Turner's examination of "The Impact of Agile and Lean on Process Improvement." However, as a frequent Crosstalk contributor, Paul Kimmerly reminds us that change created by process improvement is not always simple in "Processes are Easy,".

This issue's line-up of articles show us how far we have come through the expansion of our knowledge and the utilization of our technology to show just what we have – and can – achieve so far.

Justin T. Hill
CrossTalk Publisher

The Impact of Agile and Lean on Process Improvement

Richard Turner, Stevens Institute of Technology

Abstract. Since the early process improvement work in the '70s and '80s, our understanding of process in software and systems engineering has changed significantly. The Agile and Lean movements have made us think differently, and our processes have changed. Have our approaches to process improvement (PI) changed as well? This article discusses how Agile and Lean concepts inform process improvement approaches to address those changes.

Twenty-five years after the first version of the Software Capability Maturity Model and the original SPICE standards, and more than a century since Frederick Taylor introduced the concept of scientific management, there have been significant changes in the way we view process in software and systems engineering. Many of these are responses to a changing engineering environment that includes more uncertainty, continuously evolving systems of systems, and rapidly changing needs and technologies.

The result has been a parade of diverse and often contradictory concepts and approaches to management, development and sustainment. INCOSE's Systems Engineering Capability Model, [1] CMMI[SM], [2] and numerous ISO/IEC process standards have co-evolved with the introduction of adaptive development concepts such as the Agile Manifesto, [3] Lean principles applied to knowledge work, [4][5] model-based engineering,[6] and value-based system and software engineering, [7] Kanban, [8] Lean Startup, [9] SAFE, [10] DevOps, [11] and the Incremental Commitment Spiral Model. [12] To add additional complexity, many of the parade's participants have mixed up, split off and re-formed into myriad hybrids along the way. There have even been fundamental changes to the revered Project Management Institute's (PMI) Guide to the PM Body of Knowledge (PMBOK). [13]

As definitions and approaches have evolved, how has this parade of ideas — each with one or more corporate, academic or consultative promoters — changed the way we approach improving engineering processes in the real world? This article discusses a number of factors observed along the parade route.

Starting with a Consensus

At the 2015 International Conference on Software and System Process, I was privileged to moderate a panel that discussed whether the way we have developed systems has significantly changed our pursuit of improved processes. Lars-Ola Damm (Ericsson), Philipp Diebold (Fraunhofer IESE), Anton Keks (Codeborne), Rory O'Connor (Lero), Lee Osterweil (University of Massachusetts) and I represented various technical and governance arenas where these ideas have played both with and against each other.

The overall consensus of the panel was that the evolution of the system development environment and the way systems are developed has had a significant impact on the mechanics of process engineering and improvement. At the same time, it has strengthened the influence of a number of fundamental principles. Together, the approaches incorporate planned experimentation and rapid adaptation, increase focus on outcomes and stakeholders, and move away from organizational control and conformance.

We could easily stop there, thank the panel for a job well done, and move to another topic. However, there are a number of interesting ideas that should be addressed, generally supporting but also illuminating the panel's concise summary. For example, what do we see as the important differences between our traditional concepts of PI and these recent approaches from the Lean and Agile community?

Process Improvement Home Grounds

In our 2003 book "Balancing Agility and Discipline," [14] Barry Boehm and I identified a set of "home grounds," or characteristics associated with what the fifth edition of the PMI Guide now refers to as "adaptive" and "predictive" project management environments. At that time, there was general antipathy (to the level of religious fervor) between the proponents of traditional development processes (so-called disciplined) and proponents of newer, lighter and more adaptive (agile) development processes. In the intervening years since that publication, there has thankfully been significant rapprochement between these factions.

I believe recasting those characteristics can serve as a lens through which to consider the evolution of process improvement. Table 1 illustrates the original software development home grounds we identified.

Although this was created as a software development spectrum, it is possible to inspect the application of PI in terms of these characteristics. Table 2 presents a personal interpretation of these home grounds in terms of the process improvement activity.

Now, let's look at each of these modified characteristics more closely.

Application

This characteristic covers the type of environment where the approach is applied as well as the general goals and activities.

Much of traditional PI has been internally focused on meeting specific standards (CMM/CMMI level, ISO Certification) based on a set of goals or recommended practices that have little to do with the end product's applicability or capability. Often these are broadly stated as critical success factors and then broken down to examples of specific practices or activities. If these practices (or other equivalent practices) are included in the process and are performed, then the improvement is assumed to be accomplished. Measurements of the impact are often included in the overall management metrics and are usually evaluated within the organizational context.

Traditional PI has often depended on process expertise and best practice to codify standard processes for all personnel. This is primarily directed toward the establishment of common and certifiable practices. However, the concept of organizational standard processes may be losing its relevance in a development environment with so much uncertainty. When there is so much change

happening, having everyone use the exact same process can often result in the "Bed of Procrustes" effect where the process requires the performers to radically redefine the project, often to the detriment of the customer. [12]

There are circumstances, however, where the stability and commonality of processes are critical to the performance and certifiability of products. Safety and security standards are the poster children for having broadly accepted and conforming processes. These are the cases where PI should be carefully orchestrated and systemically controlled.

Adaptive methods echoing Agile and Lean values apply a more individualized experimental approach, whereas those performing the work are always looking for improvement opportunities. They can apply changes for improvement within every cycle, tracking the results to determine success or failure. This echoes the expressed goal of a level 5 CMMI organization. Figure 1 shows one combination of the two approaches introduced by Vic Basili under the name "Quality Improvement Paradigm." [15] This approach grew out of 25 years of PI work at a NASA facility, where a team developed models specifically around the process needs of the organization and empirically evaluated the models via experiments.

Governance

This characteristic describes how the approach is managed, resourced, and measured.

Although often initiated from the bottom up as a response to management failure, process improvement was created as a management-driven activity. Traditionally there has been a large infrastructure associated with process improvement with special groups of process experts as well as assessors and QA people that enforce standard processes. The responsibility being placed in an organization (such as an SEPG) that lies outside the performing organization does not necessarily establish ownership of the improvement with the primary performers.

This model has worked relatively well in large organizations where there was a desire for standard practices and the ability to share resources between projects without process-related learning curves. While these large organizations generally have a higher probability of sufficient overhead funding, it also led to larger PI activities with relatively long time scales for improvement evaluation.

The adaptive nature of contemporary work makes this a very inefficient way of approaching PI. Technology has incorporated and often now enforces a certain level of process (as in model-based engineering, repository-based CM tools or required software development kits). Development is seen more as brownfield evolution rather than greenfield creation, so actual development activities have realigned in importance. Concurrent engineering requires continual adjustments among hardware, software and operational concept as the system evolves. Newer approaches to PI tend to fit better in these environments, primarily because they are more highly integrated into the work. They require consistent and supportive communication among management, developers and customers.

A traditional, organizational-centric model can considerably raise the cost of PI in two ways. First, the expense of maintaining an independent or matrixed process group and a set of standard processes is not negligible. Second, whenever there is

Characteristics	*Agile*	*Plan-driven*
Application		
Primary Goals	Rapid value; responding to change	Predictability, stability, high assurance
Size	Smaller teams and projects	Larger teams and projects
Environment	Turbulent; high change; project-focused	Stable; low-change; project/org. focused
Management		
Customer Relations	Dedicated on-site customers, where feasible; focused on prioritized increments	As-needed customer interactions; focused on contract provisions; increasingly evolutionary
Planning/Control	Internalized plans; qualitative control	Documented plans, quantitative control
Communications	Tacit interpersonal knowledge	Explicit documented knowledge
Technical		
Requirements	Prioritized informal stories and test cases; undergoing unforseeable change	Formalized project, capability, interface, quality, forseeable evolution requirements
Development	Simple design; short increments; refactoring assumed inexpensive	Architect for parallel development; longer increments; refactoring assumed expensive
Test	Executable test cases define reqts.	Documented test plans and procedures
Personnel		
Customers	Dedicated, collocated CRACK* performers	CRACK* performers, not always collocated
Culture	Comfort and empowerment via many degrees of freedom (thriving on chaos)	Comfort and empowerment via framework of policies and procedures (thriving on order)
* Collaborative, Representative, Authorized, Committed, Knowledgeable		

Table 1. Home Grounds from Boehm and Turner [14]

Characteristic	*Predictive (Traditional)*	*Adaptive*
Application	**Certification or appraisal, internally focused; large, multiple team organizations with standard processes; good for highly critical processes**	**Rapid improvement, customer satisfaction, externally focused; individual teams; neutral to criticality**
Governance	**Organizational responsibility, process owners are process team (SEPG) and the organization**	**Individual responsibility, rapid evaluation of impact, process owners are the actors**
Values	**Repeatability, uniformity, satisfaction of requirements, specific process areas of activities**	**Customer satisfaction, quality of product, entire lifecycle**
Personnel	**Organizational team drives solutions, benefits often accrue mostly to marketing or management; standard practice drives knowledge**	**All personnel are involved with solutions and benefits; cross-fertilization of teams spreads knowledge**

Table 2. Interpretation of Home Grounds for the Process Improvement Spectrum

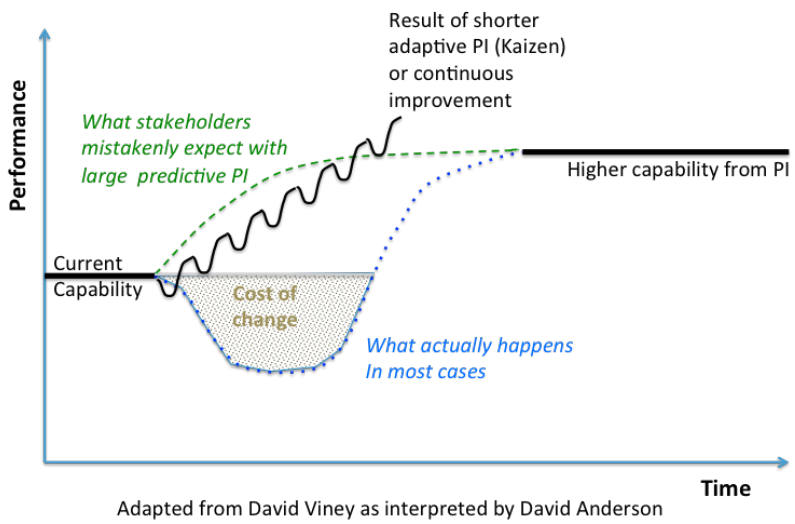


Figure 1. The Quality Improvement Paradigm [illustration from 17]

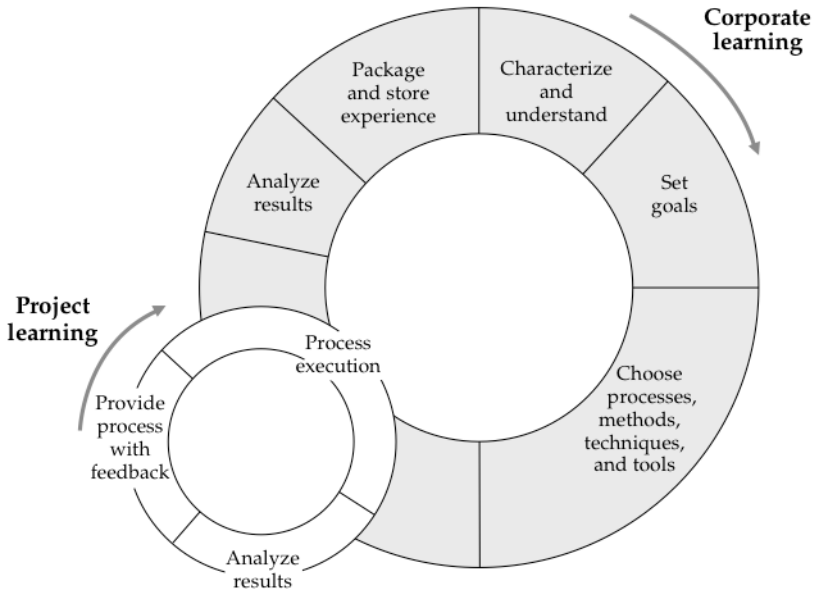


Figure 2. The J Curve (Illustration based on [16])

a "large" process improvement project, there is an unavoidable churn across organizations as the performers change and adopt the processes. This effect, often referred to as the "J curve," effectively reduces productivity for a period of time relative to the length of the process until the change is complete and the organization experiences the promised benefit. Figure 2 illustrates how the curve impacts improvement, and it shows the difference in effectiveness of shorter, continual improvement approaches.

Adopting the idea of more tailored, individually driven process improvement has been difficult, and it is particularly uncomfortable in a command and control management structure. However, it fits well into the more collaborative management approaches emerging in many companies. Rather than having additional organizational overhead, adaptive approaches are more likely to use coaching as opposed to traditional project management to help individual teams instill improvement as a part of normal work.

Values

One of the drawbacks of earlier process improvement approaches was the concept and distribution of value. The overall value of the process improvement initiative was often situational at best and nebulous at worst. Where it was seen as a necessity for competitive credibility, the value was in passing the audit rather than in any value to the organization and the customer. In other cases, the value was essentially associated with the success of one or two champions and disappeared if they failed, changed positions or left the company. On those occasions where PI was primarily instituted for the actual improvement of the organization, the internal focus on practices was often valued as a way of cutting costs, standardizing work, or deploying better predictive management capabilities rather than improving the product or raising customer satisfaction.

With the emergence of Agile and Lean, the concept of value became more aligned with outcomes. The focus on value stream and value-based decision making and scheduling brought additional considerations to what were once considered best practices. In Lean Startup and its laser focus on the market, value is intensely associated with buyers and their desires, known or unknown. Process improvement that does not improve the ability to adapt has little value.

The values of PI in adaptive environments are both holistic and individualistic. Agile and Lean focus on the ability to satisfy the customer through value delivered for cost and product suitability, and the ability to provide the individual with resources to own and benefit from improvement. Traditional PI values standardization and specifications and organizes according to key technical areas rather than the overall value chain.

Personnel

This characteristic covers a good deal of ground but essentially looks at the preferences of the people involved in the process improvement.

Traditional PI focused more on the process than on the people performing the process in a highly sterile and rigid atmosphere. Often awards went to the team leading the PI project rather than to those who suffered through the transition. The concept of having a process expert that did not really understand you and your work, but was trying to squeeze you into a predefined role or task, caused more stress than was probably necessary.

Much of the fervor behind the Agile Manifesto came from the recognition that creative people who are doing knowledge-based work are not plug-and-play resources. Even with standard processes, they operate according to soul more than to programmed instructions. People vary along the same types of spectrums as their projects, environments, and PI approaches. Thus, understanding the people following the process is key.

Lean has shown that we have outgrown Taylor's view that workers are too stupid to understand the "science" surrounding their tasks. In an age of automation and technical service industries, this point of view is rarely applicable. And, given the number of books on empowerment, coaching, collaboration and projectless workflow, it is clear that management theory is catching up.

In adaptive organizations, the team performing the work is responsible for their own processes. These team members are individually involved in both the PI activities and the derived benefits. These organizations rely on cross-fertilization of personnel across multiple projects to organically improve the organization as a whole.

A Way Forward from Looking Backward

So what is the purpose of process improvement? Returning to the consensus of the panel, process improvement techniques have certainly changed to adapt to the new realities of system development. Both approaches to process improvement follow some form of the "plan–do–check–act" or "observe–orient–decide–act" cycles for identifying barriers and enablers to improvement.

Differences in process improvement approaches seem more common in the governance and value characteristics. They echo the general changes in the development process from the greenfield, highly defined projects in the '70s and '80s to the brownfield, uncertain, rapidly evolving projects of today. Similarly, there have been more hybrid development life cycles and models to fit specific developmental and environmental needs, so

combinations of predictive and adaptive process improvement implementations have emerged.

Fundamentally, all processes and the approaches implemented to improve them should be engineered to be as amenable to change as the environment requires. Hybrid approaches are a principal means of assuring this, and their structure and content fall naturally out of a review of risks associated with the holistic environment. Ways to balance approaches using risk is described in detail in [14] and in the Meta-principle of Risk Balancing in [12].

Finally, process improvement fundamentals, generally derived from change management fundamentals, remain valid. Implementing them to PI approaches is a more difficult challenge. Here is a list of critical success factors, drawn from experience with matching PI approaches to needs in software, systems, and systems of systems evolution:

- Improve for the benefit of the business, organization, and personnel, not some externally mandated target.
- Clearly identify and exemplify the desired PI values, both internal and external, and use them to determine your approach.
- Fit the approach to the environment.
- Follow the pain in prioritizing what is to be improved.
- Understand the current capability; set achievable, measurable and meaningful goals; track progress.
- Experiment and deploy incrementally; fail fast and safely; reduce the improvement cycle time.
- Utilize reflection techniques to provide "double-loop learning": find an error, correct it, and then try to understand how it happened to prevent it in the future.
- Involve and empower the people who do the work to adapt and improve their own processes.

No list of one-liners can replace understanding, and process improvement is critically dependent on understanding the environment, the organization's values and needs, and most critically, the people. Improvement is necessary, so identifying the collection of practices that holistically best suits the improvement target worthwhile.

In the *CMMI Survival Guide,* [16] a book much more about process improvement than about CMMI, Suzanne Garcia Miller and I used the metaphor of a journey to describe our philosophy for process improvement. I think it remains both accurate and deep. My favorite epigraph included in the book sums it up nicely:

You must travel a long and difficult road –
a road fraught with peril, uh-huh,
and pregnant with adventure.
You shall see things wonderful to tell. . . .
I cannot say how long this road shall be.
But fear not the obstacles in your path,
for Fate has vouchsafed your reward.
And though the road may wind, a
nd yea, your hearts grow weary,
still shall ye foller the way,
even unto your salvation.

– An old blind man on a flatcar in
"O Brother, Where Art Thou?",
a film by Ethan and Joel Coen.

REFERENCES

1. INCOSE, "A Systems Engineering Capability Model." Vol 1. (June 1996.)
2. Chrissis, Mary Beth; Conrad, Mike & Sandy Shrum. (2011.)"CMMI for Development v1.3." Addison-Wesley, Boston.
3. Beck, K., et al. "Manifesto for Agile Software Development." http://agilemanifesto.org.
4. Reinertsen, Donald G. (1997.) "Managing the Design Factory: A Product Developer's Toolkit." The Free Press, New York.
5. Shalloway, Alan; Beaver, Guy & Trott, James R. (2009.) "Lean-Agile Software Development: Achieving Enterprise Agility." (1st ed.). Addison-Wesley Professional, Boston.
6. Friedenthal, S., Moore, A. & Steiner, R. (2011.) "A Practical Guide to SysML: The Systems Modeling Language." Elsevier.
7. Biffl, Stefan et al. (2006.) Ed., "Value-based Software Engineering." Springer, Berlin.
8. Anderson, David. (2010.) "Kanban: Successful Evolutionary Change for Your Technology Business." Blue Hole Press, Sequim, Wash.
9. Ries, Eric. (2011.) "The Lean Startup." Crown Publishers, Danvers, Mass.
10. Leffingwell, D. et al. "Scaled Agile Framework (SAFe) 4.0." http://www.scaledagileframework.com.
11. Kim, Gene et al. (2016.) "The DevOps Handbook." IT Revolution Press, Portland, Ore.
12. Boehm, B.; Koolmanojwong, S.; Lane, J. & Turner, R. (2014.) "The Incremental Commitment Model: Principals and Practices for Successful Systems and Software." Addison-Wesley, Boston.
13. Project Management Institute and the IEEE Computer Society. (2013.) "Software Extension to the Program Management Body of Knowledge Guide Fifth Edition." Project Management Institute Incorporated. Newton Square, Penn.
14. Boehm, Barry & Turner, Richard. (2003.) "Balancing Agility and Discipline: A Guide For The Perplexed." Addison-Wesley, Boston, Mass.
15. Basili, V. (Sept. 1993.) "The Experience Factory and its Relationship to Other Improvement Paradigms." In "Proceedings of the Fourth European Software Engineering Conference (ESEC) in Garmish-Partenkirchen, Germany." The Proceedings appeared as lecture notes in "Computer Science," Sept. 1993.
16. Anderson, David. (March 29, 2016.) "Organizational maturity & the J-Curve Effect." Blog post, http://anderson.leankanban.com/organizational-maturity-the-j-curve-effect.
17. Miller, S. G. & Turner, R. (2007.) "CMMI Survival Guide: Just Enough Process Improvement." Addison-Wesley, Boston, Mass.

ABOUT THE AUTHOR

Dr. Richard Turner has 40 years of experience in systems, software and acquisition engineering in both the private and public sectors. Currently a Distinguished Service Professor at the Stevens Institute of Technology, Turner is active in the Agile and Lean engineering communities and was a core author of the IEEE-CS/PMI Software Extension for the Guide to the PMBOK. His current research uses simulation to investigate the effectiveness of Agile, Lean, service and complexity concepts in managing SoS evolution. He is a Golden Core awardee of the IEEE-CS, a senior member of the IEEE, and co-author of four books.

Modernizing Earned Value Management

Wayne F. Abba, Abba Consulting

Abstract. Earned Value Management (EVM) has been part of Department of Defense (DoD) acquisition policy for 50 years, remains an essential part of that policy, and is growing internationally. EVM's longevity is discussed from the unique perspective of one who led that evolution as a public servant in the Office of the Secretary of Defense (OSD) for many years and is helping to define its newest form — Integrated Program Performance Management (IPPM).

Background

After a half century, anyone familiar with DoD acquisition policy for major programs should understand EVM principles. If not, the literature is extensive. For an excellent explanation and history, see Fleming and Koppelman, EARNED VALUE Project Management.[1] The authors traced EVM's origins back to industrial management processes from more than a century ago and noted that, as a matter of Defense policy, nothing substantive had changed in its first four decades. That remains true today.

EVM's longevity is attributable to its nonprescriptive nature and its holistic, integrative approach to industrial management. The EVM pioneers did not tell the industry "how to manage" but rather defined a set of mandatory, scalable criteria for industrial management. Those criteria, now referred to as "guidelines," have proved remarkably resilient because they relate to underlying essential management concepts such as defining, organizing, scheduling and measuring work performance.

The other key EVM attribute, integration, refers to relationships between industrial management processes and project (or contract) work. Simply put, as a contractor extends the customer's Work Breakdown Structure (WBS), EVM requires that all work is identified, budgeted and scheduled to the extent practicable. This disciplined planning makes possible the reliable measurement of project performance against a baseline and the ability to forecast the outcome.

The DoD Comptroller was the original policy owner for EVM. This proved to be a two-edged sword. While independence from engineering and acquisition disciplines allowed EVM to establish itself, the Comptroller's ownership identified it with financial management and reporting. Indeed, the first DoD EVM policy was called "Cost/Schedule Control Systems Criteria" (C/SCSC or CS^2) and was issued in 1967 as a DoD instruction in the Comptroller's 7000 series. There was an accompanying instruction for reporting.

It was many years before responsibility for EVM was transferred to the Office of the Under Secretary for Acquisition & Technology in 1989, and it was two more years before EVM was incorporated into the 5000 series in 1991.[1] With EVM having proved itself over more than two decades, the transfer placed EVM in its proper context as the essential integrating management discipline for major acquisition programs. New management processes, notably the Integrated Baseline Review (IBR), were developed to improve contract planning and execution. The DoD Acquisition Reform era of the late 1990s further served to strengthen EVM.

As with any "control" policy, EVM was not without its detractors. Through five decades it's been challenged, examined and reexamined by various auditors and reformers, always emerging stronger while other management fads came and went. OSD staff confidence in the merits of integrated project management using EVM grew as governments in other nations studied and adapted U.S. EVM techniques for their acquisition organizations.

In the mid-1990s, the Office of Management and Budget (OMB) mandated EVM for all government agencies. At the same time, OSD reached out to industry experts to develop a standard that could reduce the need for the government to define industrial management requirements. In 1998, that led to the American National Standards Institute standard EIA-748-98, "Earned Value Management Systems," issued by the Electronic Industries Alliance.[2] The criteria were virtually unchanged.

EVM gained further traction in the global project management community in 2005 when the Project Management Institute (PMI®) published the Practice Standard for Earned Value Management.[3] Thus in its first four decades, EVM evolved from a set of industrial management criteria defined by the government to a set of guidelines defined by the industry, codified in a national standard and embraced by PMI® and other professional associations.

EVM and Information Technology

The relationship between EVM and information technology (IT) has been fractious. That was not the case in the early years, when IT development was much different than it is now and typical lines-of-code measurement worked well with EVM. That changed as new techniques were developed. Shortly before the author retired from OSD in 1999, the executive in charge of IT policy met with him to discuss issues being raised by her staff.

She said some people asserted that because EVM depends on a definite scope of work, and because software engineers don't know what they will do in spiral development, the two were incompatible. This argument doesn't hold water, however, because defense contracts are not (or should not be) open-ended. Further, EVM is fully able to accommodate changes to the sequence of work and changes that revise the contractual scope of work. The executive was persuaded and EVM remained a part of the DoD's IT acquisition policy.

As years passed, the issue resurfaced occasionally. Spiral, waterfall — each new IT development technique renewed the assertion that "software is different." And that was increasingly true, at least in the commercial marketplace where requirements for products such as cell phones are not as defined as, for example, those for a developmental avionics system that must be compatible with other defense systems.

With the evolution of Agile development, the issue intensified. Several organizations began investigating the respective roles of Agile and EVM, including the Government Accountability Office (GAO), National Defense Industrial Association (NDIA), OSD and the College of Performance Management (CPM), a not-for-profit professional association that represents and advocates for EVM.

The GAO has researched Agile development as part of its ongoing project to issue a series of "best practice" guidelines. As of this writing, the research is continuing, with GAO teams having "shadowed" Agile teams at several companies and government organizations. The results will be incorporated as appropriate in the cost and schedule guides that have been published.[4] Through semiannual meetings with an expert advisory panel,[5] the GAO ensures that it is up to date on Agile and EVM developments. An example of such developments is "Agile and Earned Value Management: A Program Manager's Desk Guide," issued by OSD.[2] Another useful document, "Techniques for Integrating Agile Development Processes into Department of Defense Earned Value Management Systems," was published by the NDIA Planning & Scheduling Working Group.[3]

Through these coordinated efforts, both government and industry are continuing to modernize EVM by adapting it to the latest management developments. CPM plays an important role by providing independent, nonattribution venues for training and workshops and symposia that clarify concepts and advance the state of the art.[6]

The Future of EVM

EVM was ahead of its time 50 years ago as management philosophy, but supporting software tools were not adequate to deal with the increasing complexity and volume of management data. This placed practical limits on systems integration. Monthly reconciliation with accounting data was the norm, and reporting lagged weeks behind the accounting cutoff. Times have changed. Today's EVM systems are capable of operating in near-real time by using labor hours to manage and measure progress. This allows contractors to synchronize their EVM systems with their business rhythm by, for example, aligning EVM with weekly or biweekly schedule status reporting rather than monthly accounting cycles.

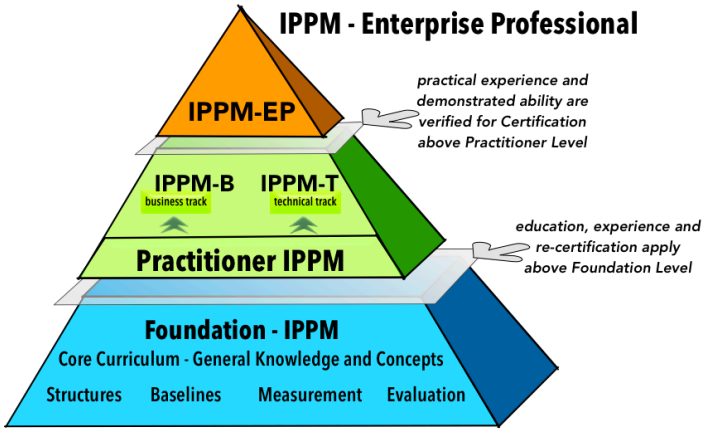


Figure 1

Given this progress, CPM is leading an initiative that draws on knowledge gained over the past 50 years to move EVM to the next level —Integrated Program Performance Management. IPPM further enhances process integration by including Technical/Benefits Management (TBM) practices. TBM prioritizes measuring and managing for results that meet business or mission needs. IPPM also emphasizes the Schedule/Resource Management (SRM) practices that are necessary to accommodate more dynamic approaches, such as Agile, to schedule planning and control methods that have emerged throughout the EVM experience.

Little exists in the way of formal education or professional credentials addressing IPPM as an integrated set of disciplines. The IPPM professional certification is emerging to fill this void in the integrated program management field. The IPPM model includes three levels of expertise — foundational, practitioner and enterprise professional. The pyramid illustration (Figure 1) gives a broad overview of the program and illustrates how practical experience and career accomplishment builds upon a knowledge base comprising the EVM, SRM, and TBM disciplines.

The IPPM foundation certification is designed to demonstrate that people have learned the general knowledge and basic concepts behind the core principles of IPPM. The intermediate (practitioner) level builds on this foundation by requiring mastery of analytical principles and ability to apply basic principles to practical settings. Applicants for the practitioner certification may choose either a "business management" or "technical management" certification to match their situation. Achieving the ultimate expert practitioner level will require both mastery of the integrated set of disciplines and evidence of practical experience and accomplishment.

Conclusion

As the senior program analyst for contractor performance measurement in OSD for nearly two decades, the author was responsible not for defending EVM, but for implementing the most effective management and measurement methods on behalf of the taxpayer. His organization's confidence that EVM was that method was confirmed as one nation after another — Australia, Canada, Japan, Sweden and the United Kingdom – adapted the U.S. model for their acquisition organizations.

The Japan experience is especially noteworthy. The nation that gave us so many management innovations — Kaizen, Deming's quality management and others — has embraced the U.S. model for integrated program management as a core function of the new Acquisition, Technology and Logistics Agency (ATLA) in the Ministry of Defense. ATLA representatives are frequent visitors to OSD, GAO, OMB and other government, industry and professional organizations as they study and adapt U.S. policies and processes.

One message they hear repeatedly is that management systems and reporting alone are not sufficient. Effective management depends on people, both in government and in industry. The systems and reports are not the end; they are a means to an end. A half-century of EVM experience has shown repeatedly that it works. It works best when both sides take full advantage of EVM and the accompanying tools that have been developed, such as the IBR and the Agile and EVM desk guide.

EVM works, whether by identifying failing contracts early and permitting timely cancellation or by facilitating timely decisions to help ensure success. Of course, the latter is preferable. History shows that the greatest successes are achieved not by having EVM specialists independently record and report on technical teams' progress, but rather by having both government and industry managers understand and use EVM effectively within a multidisciplinary team. IPPM will prepare the next generation of managers by building on the knowledge gained over 50 years on hundreds of defense programs.

ABOUT THE AUTHOR

Wayne Abba is an independent consultant in program management. He was the senior program analyst for contract performance management in the Office of the Under Secretary of Defense (Acquisition & Technology) for 17 years before retiring in 1999. He is an expert adviser to the GAO "Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Capital Program Costs" and "Schedule Assessment Guide: Best Practices for Project Schedules" and is the President, CPM.

6421 Lyric Lane
Falls Church, VA 22044
Phone (703) 658-1815
abbaconsulting@cox.net

NOTES

1. Currently DoD Instruction 5000.02. (January 7, 2015.)
2. Currently EIA-748 Revision C. (March 1, 2013.) Published by SAE International.
3. Currently 2nd Edition. (2011.)
4. "GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Capital Program Costs." (March 2, 2009.) http://www.gao.gov/products/GAO-09-3sp; and "GAO Schedule Assessment Guide: Best Practices for Project Schedules," (Dec. 22, 2015.) http://www.gao.gov/products/GAO-16-89G.
5. The author is a member of the GAO expert advisory panel and a contributor to the cost and schedule guides.
6. www.mycpm.org

REFERENCES

1. Fleming, Quentin F. & Koppelman, Joel M. (2010.) "EARNED VALUE Project Management (4th Edition)." Newtown Square, Penn. Project Management Institute, Inc.
2. "Agile and Earned Value Management: A Program Manager's Desk Guide." (March 3, 2016.) OUSD AT&L (PARCA). http://www.acq.osd.mil/evm/NewsList.shtml.
3. "Techniques for Integrating Agile Development Processes into Department of Defense Earned Value Management Systems." (October 2016.) Arlington, Virginia. NDIA Planning & Scheduling Working Group.

DO-332/ED-217 Using Modern Software Practice in Airborne Systems

Michael R. Elliott

Abstract. In civil airspace, the methods needed to produce software compliant with airworthiness have been considered overly burdensome, expensive and process-heavy. This view is based largely on experience with DO-178B/ED12B[5], which was the standard for software development in civil airspace beginning in 1992. Finalized in 2011, DO-178C/ED12C[6] was created to address these concerns and to apply more modern software practice to issues of software production and verification. Of special interest is DO-332/ED-217[7], the *Object–oriented Technology and Related Techniques* supplement to DO-178C/ED-12C, which addresses modern software practice and how it can be utilized in the production of software in systems that need airworthiness certification. This article addresses the traditional view of software development in this area and the significant cost and schedule reduction which can be realized by using the guidance and recommendations described in DO-332/ED-217 over those practices based on DO-178B/ED-12B.

I. Introduction

The formal standard *Software Considerations in Airborne Systems and Equipment Certification*, known in the industry as DO-178B or more recently DO-178C, is the means by which certification authorities, such as the FAA[1] and EASA[2], determine whether aircraft and engines containing software as part of their operational capability can be granted airworthiness certification for operation in civil airspace. As such, it is required reading for thousands of engineers worldwide who produce software for aircraft and aircraft engines. It specifies the means by which such software is produced and verified so that airworthiness certification can be granted.

Military aircraft, such as the USAF C-17, have made use of DO-178B for guidance in airworthiness even though not formally required to do so. An effort began in late 2004 to produce a successor document to DO-178B/ED-12B to be known as DO-178C/ED-12C. Special Committee 205 (SC-205) of the RTCA[3] and Working Group 71 (WG-71) of EUROCAE[4] were formed to address the perceived shortcomings of the existing standard from a viewpoint more attuned to modern software practice.

The mission of this subgroup was to address the needs of software practitioners in creating object-oriented software for airborne systems, which was a practice widely viewed as prohibitively difficult under the existing standard. The core document of DO178B/ED-12B was changed to help facilitate this effort. A supplement to the emerging DO-178C/ED-12C was produced, providing additional (and sometimes alternative) objectives, guidance and recommendations to aid the practitioner in airborne software production and the certification authorities in approval processes.

A. Object-Oriented Technology (OOT)

To date, few airborne computer systems in civil aviation have been implemented using OOT. Although OOT is intended to promote productivity, increase software reusability, and improve quality, uncertainty about how to comply with certification requirements has been a key obstacle to OOT use in airborne systems. (OOTiA[1], 2004.)

The importance of object-oriented technology was recognized as a key element to be addressed. One of the three subgroups formed to address software development practice was Subgroup 5 – Object-oriented and Related Technologies.

B. Object-oriented Technology Supplement

This supplement, as IP[5] 500, was formally approved at the SC-205/WG-71 plenary session in Paris, France, on Oct. 29, 2009. It is singularly appropriate that the day that the last-ever OOPSLA[6] ended – the day that object-oriented programming was considered so mainstream that it was no longer worthy of a special conference – is the day that marked the first formal acceptance of the use of object-oriented programming in the international standards for safety-critical airborne software.

II. Background

Initially, software was viewed as a way to inexpensively extend the versatility of analog avionics. However, software in the system did not fit easily into the safety and reliability analysis based on mean time between failure and other service history based techniques.

A. DO-178

This initial effort at a standard for software development in airborne systems was a set of best practices that was created to provide a basis for communication between applicants and certification authorities. It required applicants to meet "the intent" of DO-178 without giving specific objectives or significant guidance on how to do so. It did, however, introduce a three-tiered system of software criticality – critical, essential and nonessential – and set the level of verification to reflect the criticality level. Additionally, it provided a link between software verification and FAA documents, such as Federal Aviation Regulations and Technical Standard Orders.

B. DO-178A

After the initial experience with certification using DO-178, there was a consensus that it needed revision. SC-152 of RTCA created DO-178A in 1985, and it turned out to be quite different from DO-178. It introduced rigorous requirements for software processes (based on the waterfall method), software production, and process documentation and history. Applicants and certification authorities

frequently misinterpreted certification artifacts, sometimes causing entire software development efforts to be abandoned. In general, knowledge of why the certification requirements existed and their purpose failed to be understood or appreciated. [2]

C. DO-178B

The avionics industry became more and more software-oriented during the time DO-178A was in use. Many new companies entered the field and produced equipment subject to certification. Lack of experience, documentation, and understanding of the reasons for satisfying DO-178A brought about a need for an improved standard. In 1992, this became DO-178B, which was developed in cooperation with EUROCAE as ED12B by SC-167 and WG-12. This updated document made many fundamental changes to its predecessor. Salient among these was the introduction of software criticality levels A through E, which replaced the "critical, essential and non-essential" designations that were used previously. It placed a strong emphasis on requirements-based testing, which was seen as a more effective verification strategy than traditional white-box testing. It also required that these tests and their related artifacts be made available to certification authorities for use as part of their approval process.

D. OOTiA

During the eight years following the release of DO-178B/ED12B and its adoption by the industry, some people expressed concern that more modern software practices were difficult to employ using that standard. In 2000, the FAA responded by contacting the representatives of several key companies, including Boeing, BF Goodrich and others, to produce an analysis of how object-oriented software procedures could be adapted to the needs of airworthiness certification.

This process was later available to the industry in general, and workshops were held in order to produce position papers that would hopefully evolve into a best practices guide, become an FAA Advisory Circular, or be rolled into the not-yet-begun DO-178C effort. The FAA and NASA[7] held meetings that eventually resulted in the FAA publishing the four-volume *Handbook for Object-Oriented Technology in Aviation* (OOTiA). This document was never intended to contain objectives or guidance for practitioners and certification authorities. It was only meant to contain a set of suggestions for best practices and warnings about problematic situations.

By 2005, the FAA had decided that it would no longer maintain sponsorship of OOTiA or facilitate any updates or corrections to it. SC-205 of the RTCA was under consideration as a means to upgrade DO-178B [1], and it was considered best to turn OOTiA over to the nascent SC-205 to use as input for the creation of an object-oriented supplement to the new standard.

E. Rationale

The views of a number of stakeholders, including certification authorities, airframe manufacturers, and equipment suppliers, were taken into account in the creation of DO178B/ED-12B. A basic tenet of this document was that it should be written, as much as possible, to be requirements-oriented; that is, the document should be about objectives rather than processes. This was fundamentally meant to minimize the impact of technological evolution, as long diatribes such as the best use of blank COMMON blocks in FORTRAN were considered inappropriate in the long term. This brought about the philosophy of creating the document in terms of objectives, guidance, and guidelines so that applicants could use it in creating airborne software and certification authorities could use it to judge software's suitability in an airworthiness determination.

A large part of the document is concerned with how software is produced, how source and object code is traced to requirements, how the requirements trace to source and object code, and how the software is tested and shown to have been adequately tested.

F. Software Certification

There is a perception among those new to this field that software is somehow "certifiable" for airworthiness. This may come from a simple reading of the title of DO-178C/ED-12C *Software Considerations in Airborne Systems and Equipment Certification.* However, software is not actually "certifiable." Entities for which airworthiness certification can be granted are aircraft, engines, propellers, and, in the U.K., auxiliary power units. This means that the effort expended on achieving the "certifiability" of software is in actual practice expended on ensuring the certifiability of the aircraft, engine, or something else that is subject to the airworthiness certification effort — not the software involved.

For example, it is not possible to produce a "certified" version of a real-time executive or garbage collector, regardless of any statements in the marketing material of a particular vendor. What software vendors may do — and typically charge a substantial fee for — is provide the requirements, source traceability, requirements-based tests and test results for a particular software component that it provides. This documentation can then be submitted to the certification authority as part of the applicant's request for airworthiness certification of an engine, aircraft, or something else.

G. Software Production Process

The DO-178 series of documents are widely perceived as process-heavy; that is, they impose a substantial burden on the applicant to show that a particular process has been followed in the production of and verification of the airborne software that is being considered for certification. Although this has been widely considered a very expensive activity, over 35 years of airborne operations have not revealed any major safety flaws. Contrast this with, for example, the maiden flight of the Ariane 5 (Flight 501, June 4, 1996), which was destroyed 37 seconds after launch due to a software coding flaw — the failure to handle an exception raised during the initial boost phase. The Ariane 5 was never subjected to a civil airworthiness certification effort as its flights through civil airspace fall under a different authority, but it serves to illustrate that software coding errors can cause spectacular disasters.

Nevertheless, many people in the airborne software industry still believe that DO-178B/ED-12B made using less process-heavy techniques, such as model-based development, formal

methods and object-oriented programming, difficult when certification aspects were considered. The attitude is often one of "We already know how to create certifiable software the old-fashioned way. Why should we change now?" There is, therefore, a substantial perceived risk to adopting more modern techniques, regardless of the reduction in cost, errors, and time to market.

III. Rationale for Change

The answer to the question above is that the cost of doing things the old-fashioned way is becoming prohibitive. It now costs hundreds of millions of dollars for a new large aircraft to achieve airworthiness certification. That makes even small increases in efficiency lead to a competitive edge for airframe manufacturers and their equipment suppliers, who are able to be more efficient in their software production. Object-oriented programming is one way substantial increases in efficiency can be achieved, if only it can be used in an approved airworthiness certification effort.

Additionally, the software world has changed. Back in the 1980s, almost all airborne software was written from scratch to run on a single processor. This was a big problem for "commercial off the shelf" (COTS) software, as it was almost certainly not developed in an airborne software environment and therefore didn't have all the traceability and requirements-based test artifacts needed for eventual certification. This, obviously, has an impact on cost.

As far as safety is concerned, there's a real benefit to investing substantial resources into doing certain things right in a project-independent manner. Consider the wisdom of using a memory management system written by a specialist in real-time garbage collectors and used by thousands of developers rather than a pooled memory system written by a specialist in terrain avoidance and used by fifteen developers.

In the 1980s, people wanted to achieve safety goals through testing. One particular objective of software testing is "to demonstrate with a high degree of confidence that errors which could lead to unacceptable failure conditions, as determined by the system safety assessment process, have been removed." [5] The realization that this objective is, by and large, unobtainable in modern software systems has gained substantial consensus. It is widely felt that this view does not scale to the complex systems of current airborne software — let alone future systems — due to both hardware and software complexity. That is, exhaustive software testing will not reach the desired conclusion that all necessary errors "have been removed." This, in turn, has brought about a refocusing of the testing effort toward more realistic goals like reaching a reasonable level of confidence that the software is correct, safe, and useful rather than completely error-free.

IV. Changes with DO-178C/ED-12C

The creation of DO-178C/ED-12C brought about five auxiliary documents:

- DO-278A/ED-109A: *DO-278A Software Integrity Assurance Considerations for Communication, Navigation, Surveillance and Air Traffic Management (CNS/ATM) Systems.*
- DO-330/ED-215: *Software Tool Qualification Considerations.*
- DO-331/ED-218: *Model-Based Development and Verification Supplement to DO-178C and DO-278A.*
- DO-332/ED-217: *Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A.*
- DO-333/ED-216: *Formal Methods Supplement to DO178C and DO-278A.*

The remainder of this article will focus on only one of these: DO-332/ED-217[7], which was meant to address coding and verification issues.

Subgroup 5 — Object-Oriented and Related Technologies — took on the challenge of addressing, to a large extent, all coding issues. While issues such as dead and deactivated code, inlining, ad-hoc and parametric polymorphism are not particularly object-oriented, subgroup 5 addressed those issues along with more obviously Object-oriented (OO) topics such as inheritance, class hierarchy consistency, and run-time polymorphism.

The overall aim was to provide clarification of objectives from an OO viewpoint, to provide any new objectives that were deemed beneficial to airborne safety, and to provide guidance and recommendations for achieving those objectives.

A. OOTiA, CAST, FAA and EASA

One of the initial responsibilities of the subgroup was to address all issues raised in OOTiA and either incorporate them into the supplement or deem them inapplicable or unfounded. IP 508 was produced by the subgroup to respond to each individual concern raised by OOTiA. Additionally, concerns about OO had been raised through CAST papers, EASA CRIs[9] and FAA IPs. The subgroup was to also address all of these.

B. Dead and Deactivated Code

DO-178C/ED-12C disallows dead code, which is basically code that can never be executed. Dead code is treated as a software error that should be eliminated. A variant on this is deactivated code, which might be executed for a particular configuration not used in flight. An example of this might be a software-controlled radio, which includes code to control a military hardware encryption/decryption device but which would not be selected for a purely civilian application. This is already addressed by DO-178C/ED-12C. However, when reusing software components — especially externally developed software components such as class libraries — this comes into play as the abstraction for a component may include more behavior than is actually exercised by the airborne software.

Consider a stack class which is used as a previously developed component and which contains methods for "push", "peek" and "pop." All of these methods fit the abstraction for how a stack should work and are not out of place in a stack class. The particular airborne software using such a stack, however, might not actually use the "peek" method. The previous standard would have forced the practitioners to actually remove the code for the peek method before certification as it would be considered dead code. The new standard relaxes restrictions on separately developed components and allows this stack class to be used unmodified.

C. Type Theory

Early on, the subgroup decided to provide a type theoretical basis as a rationale for reducing the amount of redundant testing and verification that involved base classes and their derived subclasses. A great deal of this testing and verification can be shown to be redundant and therefore unnecessary through type-theoretical arguments that involve class hierarchy design, as long as the type hierarchies in question share certain properties. There is a notable absence of type theory — or, for that matter, any sort of formal computer science — as a basis for decision-making in DO-178B/ED12B. Subgroup members perceived this as being at some risk of being rejected by the subcommittee as a whole, but it was accepted in the end.

D. The Liskov Substitution Principle

This sort of type-theoretical formulation initially manifested itself in the specification of the Liskov Substitution Principle (LSP) [3] as the basis for establishing that superclass behavior verification could be used as part of the verification compliance of the subclass of that superclass. The point was that only the additional behavior provided by the subclass needed to be verified if that subclass conformed to LSP. Consider the formulation LSP which appears in the supplement:

Let *q(x)* be a property provable about objects *x* of type *T*.

Then *q(y)* should be true for objects *y* of type *S* where *S* is a subtype of *T*.

Regardless of the succinctness of this, the subgroup felt that a purely theoretical expression of this concept might place too great of a burden on the practitioners.

1) Explaining LSP: As the supplement neared completion, members of subgroup expressed that the definition given above needed to be explained more clearly. The inclusion of a Frequently Asked Questions (FAQ) section in the supplement provided a less structured environment into which the subgroup could place questions and answers that were presumed to be destined to be frequently asked. These essentially addressed the question "What's the deal with the Liskov Substitution Principle and why should I care?" The subgroup could have simply reiterated the concept and continued to claim that it was a good thing — which is true — but that probably wouldn't get the point across. Based on the idea that seeing a car crash is more conducive to reminding drivers why safety is important than listening to safety lectures, it was decided to show how failure to follow LSP could lead to problematic behavior. DO-178C/ED-12C is fundamentally a document about software safety, so this approach was considered reasonable.

2) Creating a Counterexample: For purposes of the supplement's FAQ, the following situation was proposed: There exists a conceptually abstract hardware speed controller that can be instantiated with the necessary behavior to reflect the hardware of many different manufacturers. This provides the necessary basis for creating a base class so that concrete subclasses could be created for each manufacturer's particular version with whatever device-specific low-level hardware interface was necessary. Additionally, some members felt that this example was something that practitioners would see as vaguely similar to the sort of software they were developing — software to control a pump for a fuel control system, maybe. A stretch, perhaps, but not an outrageous one.

3) Preconditions, Postconditions and Invariants: The argument is made that a number of different manufacturer's speed controllers would be substitutable for the base class as long as they correctly implemented the adjust speed method to communicate the desired increase in speed through whatever hardware-specific means necessary. A class invariant for the speed controller is that an instance's speed attribute is the magnitude of the velocity and therefore can never be less than zero. To use the Java terminology, the base class creates a means to adjust the speed by giving a speed increment to an adjust speed method. This adjust speed method's postcondition is that when given a positive, nonzero argument, the speed attribute of the object has increased. Therefore, it must be nonzero.

4) Time to Divide by Zero: Based on this postcondition and invariant, a method "time to go," taking a distance argument, will return in whatever units are convenient the time value it takes to traverse that distance. This ultimately reduces to dividing the given distance by the object's current speed attribute, then converting it to the correct units. The situation as outlined above represents a valid use of LSP. Any desired number of subclasseses of the speed controller can be created, each of which tailors its behavior to what is required by the underlying hardware. In order to demonstrate the failure of LSP, the subgroup introduced an "auto controller," a subclass of speed controller designed to control a fundamentally different type of hardware that is given a desired speed that it seeks to reach and maintain.

5) Breaking LSP: Since this new auto controller class no longer needs the adjust speed by a speed increment method, its necessary implementation of the method does nothing. Additionally, a "set desired speed" method would need to be introduced to address the new abstraction of this type of speed controller. The point of all this is that by having the auto controllers adjust speed method do nothing, the postcondition is violated, since invoking the adjust speed method on an object with zero speed would fail to make the actual speed attribute nonzero. This, in turn, would cause division by zero when the "time to go" method was invoked, causing a division by zero exception to propagate through the system. This should leave the reader with an image of the smoke and debris cloud ultimately resulting from that unhandled exception on the Ariane 5's maiden flight.

A. Local and Global Class Hierarchies

The supplement includes a brief explanation of the concept of hierarchical encapsulation so that it could form the basis for a discussion of class hierarchies which, in turn, brings about a discussion of type consistency for local and global type hierarchies. The supplement uses the term "local type consistency" to provide a means to determine type consistency in a component, independent of the type consistency of code which might utilize that component. That is, developers could make type consistency determinations with well-defined boundaries, facilitating the incorporation of separately (and often externally) developed class hierarchies.

B. Taxonomy of Polymorphism

Although not really object-oriented in nature — the charter of the subgroup being essentially all coding issues — the notion of polymorphism is approached from a type-theoretical basis as well. With a brief description of the forms of polymorphism as being universal polymorphism and ad-hoc polymorphism, each of these is discussed as being divided into parametric and inclusion polymorphism, coercion and overloading, respectively. Again, the subgroup did this with some apprehension but felt that at least introducing the vocabulary would provide additional means of clarifying situations where polymorphism is used and provide a common vocabulary for practitioners and certification authorities. A similar philosophy guided the decision to discuss closures as a means of specifying behavior; that is, if the terms are introduced in the supplement, an applicant can use the concept with an expectation that the certification authority will at least be on the same page.

1) Resource Management: One area in which DO-332 expects to have a large impact on software design in airborne systems is the provision of a section on resource management, especially heap management, where automatic garbage collection is explicitly permitted for the first time. Garbage collection in real-time systems is a subject on which a great deal of religious fervor has been expressed in the software safety community, especially the ongoing theme that garbage collectors are somehow too complex and therefore should not be allowed in a real-time or safety-critical situation.

A consistent problem encountered with this view is the inability of any of its proponents — or at least the ones with whom the subgroup communicated — to express just how complex "too complex" is, or even how such complexity should be measured. It was found to be especially curious that the notion was expressed — and fiercely defended — that garbage collectors were inherently too complex to be used in aviation but that high-bypass turbofan jet engines somehow were not.

While rejecting the notion that garbage collection — now and, presumably, forever — is unusable due to some unspecified and undefinable algorithmic complexity in all garbage collectors, the supplement recognizes the potential for heap memory exhaustion in an airborne system and gives guidance to detect it and provide a degraded mode into which the subsystem can transition if such a situation becomes imminent. The idea of throwing an unhandled out-of-memory exception is still possible, just as is the throwing of an unhandled division by zero exception. But the guidance and recommendations give developers and certification authorities a specific set of criteria to verify.

2) Functional Programming: There is an expectation that functional programming will be an increasingly important technique in future embedded systems, including airborne systems. As the future of processors seems to be in more cores rather than in more speed per core, a widely accepted technique for handling additional throughput is to move to a computation model involving an increased use of concurrency.

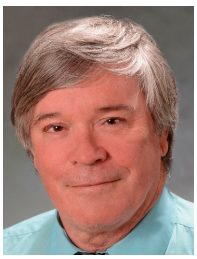
Handling concurrency will be increasingly important, and a well-established method for handling massive concurrency is the use of immutable data and functional programming. This was part of the driving force behind providing guidance for polymorphism, closures and garbage collection, all of which are heavily used in functional programming. The subgroup wanted to ensure that it left a path to functional programming for future practitioners.

V. Conclusion

Generally, the subgroup took the view that it should strive to remain language- and technology-neutral, but that it should use real languages (Ada, C++ and Java, in particular) and technology to provide examples and illustrations of problem areas (for example, static dispatch and violation of the Liskov Substitution Principle). The subgroup also subscribed to the view that this supplement will be the foundation for perhaps two decades of future safety-critical software implementation, so the subgroup needed to be careful and conservative in the resulting document. This was also done with the knowledge that the real-time, avionics and safety-critical communities are reluctant to introduce new concepts (garbage collection and runtime polymorphism, for example), so the subgroup needed to provide a basis for acceptability of such ideas to that community by furnishing a theoretical base for discussion as well as an analysis of the perceived risks of a given approach and recommendations for mitigating those risks.

In general, the subgroup feels that DO-332/ED-217 provides a sound collection of techniques to mitigate the difficulty and expense involved with creating and validating airborne software, now and in the future.

ABOUT THE AUTHOR



Michael R. Elliott is a software engineer with a deep passion for modern software practice, embedded systems architecture and safety- and security-critical software. Elliott was a member of SC-205/WG-71 Subgroup 5, which developed DO-178C/ED-12C and DO-332/ED-217. He has a bachelor's degree in information and computer science from the University of California, Irvine and a master's degree in software engineering from Edinburgh University, Edinburgh, Scotland.

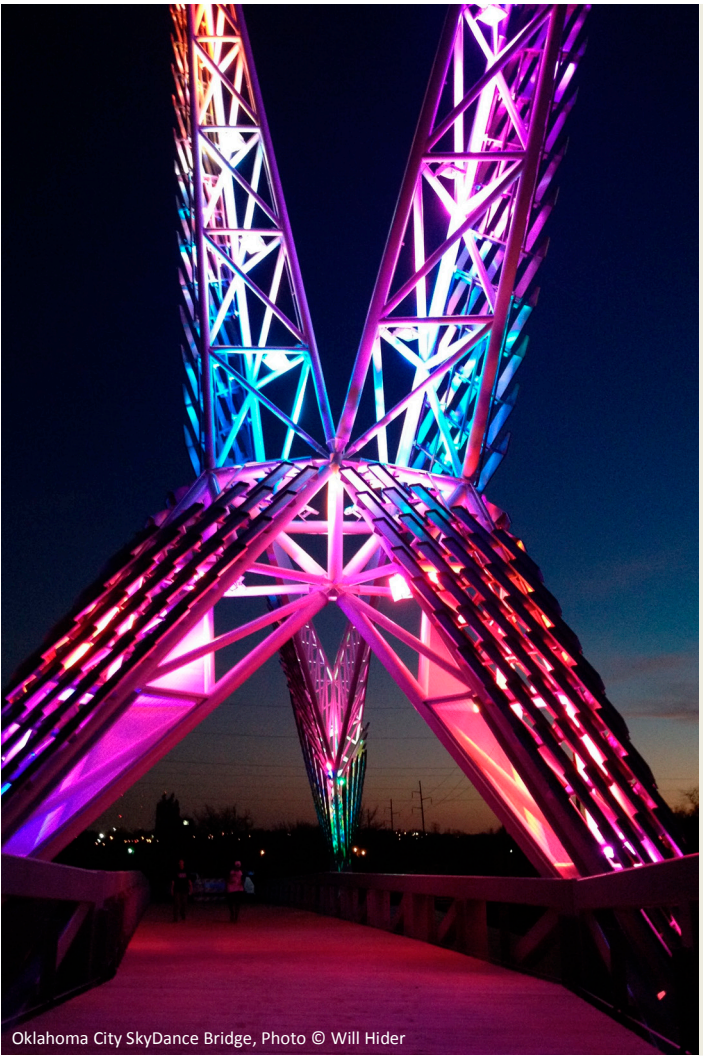
aicas GmbH
Haid-und-Neu Straße 18
76131 Karlsruhe Germany
elliott@aicas.de

NOTES

1. Federal Aviation Administration. Washington, D.C., USA. http://www.faa.gov.
2. European Aviation Safety Agency. Cologne, Germany. http://www.easa.europa.eu.
3. RTCA, Inc. (Washington, D.C., USA) is an organization that creates standards documents for the FAA. http://www.rtca.org.
4. The European Organization for Civil Aviation Equipment (Malakoff, France) is an organization that produces documents referred to as a means of compliance for European Technical Standard Orders. http://www.eurocae.net.
5. Information Paper.
6. Object-Oriented Programming Systems, Languages and Applications conference of the Association for Computing Machinery.
7. National Aeronautics and Space Administration. Washington, D.C., USA. http://www.nasa.gov.
8. Certification Authority Software Team, a group of individuals representing several certification authorities, including EASA, FAA, JAA and Transport Canada.
9. Certification Review Items.

REFERENCES

1. Federal Aviation Administration (FAA). (Oct. 2004.) "Handbook for Object-Oriented Technology in Aviation (OOTiA)." Washington, D.C., USA.
2. Johnson, L. (Oct. 1998.) DO-178B, "Software Considerations in Airborne Systems and Equipment Certification." Crosstalk, The Journal of Defense Software Engineering, 11 (10).
3. Liskov, B. & Wing, J. (Nov. 1994.) "A Behavioral Notion of Subtyping." ACM Transactions on Programming Languages and Systems, 16(6): 1811–1841.
4. Special Committee 152 of RTCA. (March 1985.) "DO-178A/ED-12A – Software Considerations in Airborne Systems and Equipment Certification." RTCA and EUROCAE. Washington, D.C., USA and Paris, France.
5. Special Committee 167 / Working Group 12 of RTCA and EUROCAE. (Dec. 1992.) "DO-178B/ED-12B – Software Considerations in Airborne Systems and Equipment Certification." RTCA and EUROCAE. Washington, D.C., USA and Paris, France.
6. Special Committee 205 / Working Group 71 of RTCA and EUROCAE. (Dec. 2011.) "DO-178C/ED-12C – Software Considerations in Airborne Systems and Equipment Certification." RTCA and EUROCAE. Washington, D.C., USA and Malakoff, France.
7. Subgroup 5 of Special Committee 205 / Working Group 71 of RTCA and EUROCAE. (Dec. 2011.) "DO-332/ED-217 – Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A." RTCA and EUROCAE. Washington, D.C., USA, and Malakoff, France.



Oklahoma City SkyDance Bridge, Photo © Will Hider

Framework for Selecting the Preferred Networked Computer System for Dynamic Continuous Missions

Glenn Tolentino, Dr. Jeff Tian, Dr. Jerrell Stracener

Abstract. This paper presents a framework for selecting a combination of existing systems to satisfy new, emerging requirements while reusing existing and proven capabilities to ensure mission success. Decision attributes will be considered during the selection process and will be used to measure the networked computer system's effectiveness to accomplish the mission. This approach will enable system stakeholders to make critical, well-informed decisions to address the continuing evolution of missions, threats, budget and technology.

Introduction

Defense Computer Systems developed and maintained over the years have resulted in thousands of disparate, compartmented, focused, and mission-driven systems that are utilized daily for deliberate and crisis mission planning activities. The defense acquisition community is responsible for the development and sustainment of these systems over the course of its systems engineering life cycle from conception to utilization and eventually to the decommissioning of these systems. In addition to the cost of these systems' acquisition and development phases, there are associated investment costs that are necessary to sustain these computer systems over their life cycles. While missions are being planned and satisfied by existing computer systems, there are new missions being proposed which cannot be satisfied by a single existing computer system capability. Therefore, this raises the question of whether a Networked Computer System (NCS) is preferred in order to satisfy new capability requirements by using combinations of existing and developmental computer systems. This paper explores an approach to identifying a preferred NCS solution and measuring the NCS's effectiveness in satisfying a mission.

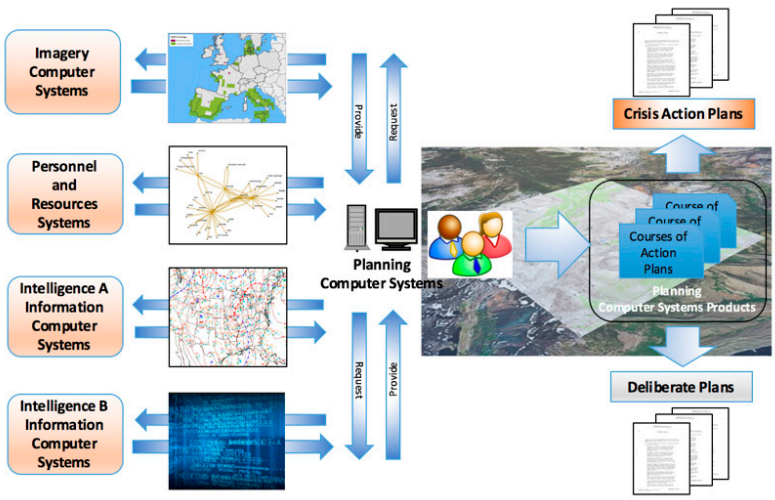


Figure 1. Notional NCS Concept of Operations

Defense Computer Systems

The United States Department of Defense (DoD) requires new capabilities with new requirements to address the continuing evolution of missions, threats, budget and technology. These new capabilities can be satisfied with existing operational systems, the development of new functionalities into existing systems, or the development of completely new systems. However, another approach is to develop a combination of existing systems for emerging requirements to satisfy new capabilities while reusing existing and proven capabilities with the goal of ensuring mission accomplishment. This "identification and selection" approach will reduce the risk associated with system development and integration efforts while providing a better-informed decision making process in satisfying user requirements while considering cost, schedule, and program execution performance. In addition to the decrease of defense spending, the acquisition community must also seek innovative ways to satisfy the new systems capabilities needed in order to accomplish the DoD operational mission.

The research performed in this paper was based on the United States DoD's need for an operational system capability that can satisfy a defense mission, and specifically seeks to determine if the capability requires a group of computer systems to be developed into a single NCS solution (see Figure 1). Once a preferred NCS solution is identified, the question becomes "How do we measure the effectiveness of these developed and integrated computer systems with the end of goal of satisfying mission success?"

Overview of Methodology Framework

The notional conceptual methodology includes a number of steps that must be accomplished in order to select computer systems in developing the NCS solution and measuring the solution's effectiveness. Table 1 describes the two phases followed by a summary describing each of the steps under each phase.

Phase 1: Selecting Computer Systems for an NCS Solution

This first phase focuses on developing an NCS solution for a given mission based on mission requirements and objectives. This phase will address the development of a NCS solution based on existing computer systems that are either already operational or currently being developed with a known time for capability readiness and acquisition. The following sections describe each of the steps.

Phase 1	
Selecting Computer Systems in Developing an NCS Solution	
Step 1	Describe the NCS Mission
Step 2	Identify computer systems with capabilities to satisfy mission required functionalities
Step 3	Determine computer systems for NCS consideration • System capability availability • Capability readiness • Acquisition time • Acquisition cost
Step 4	Determine NCS solution to satisfy mission
Phase 2	
Determining the Measure of Effectiveness of the NCS solution	
Step 1	Evaluate NCS solution based on decision attributes: Capability sustainment definition and estimation Mission reliability definition and estimation Lifecycle cost definition and estimation
Step 2	Determine measure of effectiveness based on decision attributes

Table 1. Methodology Framework Phase One and Two

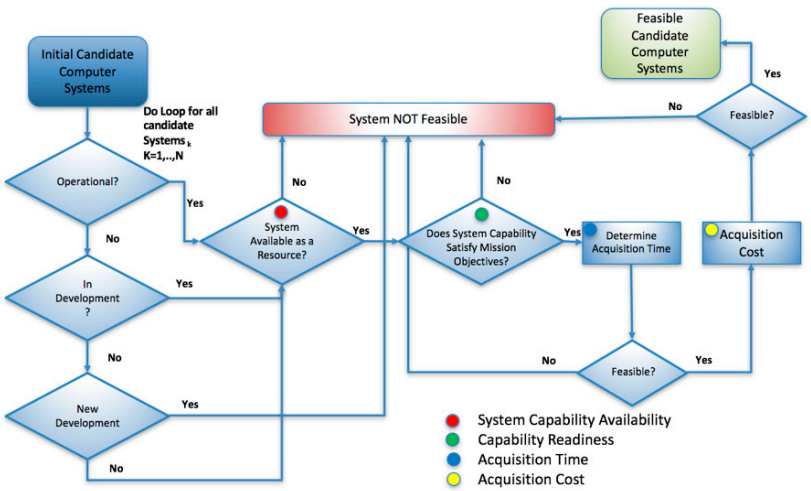


Figure 2. Approach for Determining Feasible Candidate Computer Systems

Phase 1 - Step 1: Describe the NCS mission:

This step describes the intended overall mission or missions of the NCS. Defining the mission is a high-overview activity that specifies what is to be performed with specific mission objectives. These mission objectives can be translated as a set of activities that must be performed in order to achieve mission success and can be characterized as the mission profile, which translates to specific capabilities. There are a number of capabilities that are required by the NCS in order to satisfy each of the mission objectives and accomplish mission success.

Phase 1 - Step 2: Identify Computer Systems to Satisfy Mission-Required Capabilities

During this step, each of the capabilities required for the NCS solution is identified. Once all of the capabilities are identified, the capabilities objectives are established along with high-level capabilities requirements to satisfy the objectives. The capability requirements are then used to determine whether initial candidate computer systems will be able to satisfy the requirements.

Phase 1 - Step 3: Determine Computer Systems for NCS Consideration

This step in the process assists in determining which computer systems will be under consideration to be part of the NCS solution. It provides a process to help select the systems based on system capability availability, capability readiness, acquisition time, and acquisition cost (see Fig. 2). This process provides a library list of computer systems for each of the capabilities required for the NCS solution.

Phase 1 - Step 4: Determine NCS Solution to Satisfy the Mission

During this step, a library list of computer systems that satisfies each of the capabilities defined by the high level capability requirements will be available as part of a down select process. This step will identify potential computer system candidates to be considered into the NCS solution. The identification process will utilize a process to determine which computer systems are the "best" candidates in accomplishing the NCS capability objectives. A selection process enables the stakeholders to be able to provide a level of balance between objective and subjective decision-making in selecting the computer systems as a component of the preferred NCS solution.

Phase 2: Determining the Measure of Effectiveness of the NCS Solution

The purpose of Phase 2 is to evaluate the NCS solution based on the decision attributes in quantifying the NCS solution's effectiveness. This phase will evaluate the NCS solution based on the decision attributes selected (capability sustainment, mission reliability, and life cycle cost) and measure the effectiveness based on the estimation. The following sections describe each of the steps of Phase 2.

Phase 2 - Step 1: Evaluate NCS Solution Based on the Decision Attributes

The NCS solution will be evaluated based on the decision attributes that are related to the Measure of Effectiveness (MOE) construct. In terms of MOE, the NCS solution will consider capability sustainment (basic reliability), mission reliability, and capability life cycle cost. Each of the decision attributes will be

quantitatively estimated and analyzed in determining the MOE of the NCS solution that could be further analyzed and evaluated.

a) Capability Sustainment Definition and Estimation:

Capability sustainment translated as basic reliability is considered to be a measure of sustainability and operations and support of a system. As defined in MIL-STD-785B, [5] "the measures of basic reliability such as Mean-Time-Between-Failures (MTBF) include all item life units (not just mission time) and all failures within the item (not just mission-critical failures of the item itself)." Basic reliability requirements apply to all items of the system.

In terms of computer systems, two primary components can affect basic reliability: software and hardware. The interrelationship between hardware and software is a primary driver that can affect the overall reliability of the system. The hardware's reliability would consist of all hardware elements of the system in terms of failure that are assessed based on failure rates of the hardware configuration items. [7] Similarly, software reliability can also be characterized in terms of the number of software components and its reliability based on the number of software failures that occur over time. As part of the informed decision making process, both hardware and software reliability and their dependencies must be mathematically formulated in order to estimate and calculate the overall reliability of the system.

b) Mission Reliability Definition and Estimation

"Mission reliability" is defined as the estimate of the probability the NCS will perform its required functions during the mission over a certain time period. This definition is based on the assumption that all mission essential items are ready and operational at the start of the mission. Furthermore, mission reliability is a system-level reliability metric that is a function of (1) the mission definition in terms of mission essential functions by mission phase and (2) the configuration and failure rates of the NCS essential items by mission phase. The mission must be defined and described in terms of the duration of each phase and the functions that must be accomplished for the NCS' mission success. The assurance of mission reliability can be attributed to systems with increased levels of redundancies and failovers. However, increasing the probability of mission success by improving the mission reliability affects basic reliability in the form of increased logistics overhead to include support, maintenance and costs. Therefore, there is an underlying dependency between basic and mission reliability considered as part of this research.

c) Life Cycle Cost Definition and Estimation

One of the requirements in the development of systems that are managed and operated by the DoD is a determined cost of its life cycle. [14] Systems developed within the defense acquisition model follow a cost model to support the affordability among all the phases of a system's life cycle to include material solution analysis, technology development, engineering and manufacturing development, production and deployment, and operation and support. [6] It is important to know the program's cost at particular intervals in order to ensure that adequate funding is available to execute the program according to plan. [11] "Affordability must be a performance consideration from beginning throughout the life cycle." [8] Similarly, the NCS solution will also consider a cost model as a measure of affordability in support of the NCS life cycle to satisfy a mission. (See Figure 3.)

Since the NCS solution will only be acquiring existing systems that are in development or systems that have already achieved their initial operating capabilities, the NCS solution will support two cost model components, (1) cost model for each of the constituent computer systems and (2) cost model for the NCS solution. [6] The first component is the costs associated with acquiring and engineering the computer systems specifically in developing, integrating, testing and deploying. These are cost drivers that involve engineering efforts for each of the computer systems that are part of the NCS solution. The second component is the costs associated with managing, utilizing, maintaining and supporting the NCS during its operational life cycle. The cost is a reoccurring cost throughout the NCS life cycle for as long as the operators utilize the NCS solution.

The cost structure and its elements are cost drivers in developing and sustaining an NCS solution throughout its life cycle. These cost drivers can be categorized by the life cycle phases of an NCS solution in the following cost structure elements table:

Phase 2 - Step 2: Determine Effectiveness Based on Decision Attributes

In this step, the NCS solution and the estimated decision attributes will be used to determine the MOE. The previous section determines the decision attributes based on a quantitative approach for measuring the attributes considered to be critical components of the MOE of the NCS. The question is how to balance all of the decision attributes that are considered important to determining a specific measure in determining the MOE of the NCS solution. Since this notional conceptual methodology is based on determining a solution to be considered based on specific decision attributes to calculate the effective measures of the system, the methodology will consider a process that is

Figure 3. NCS Life cycle

Phase Number	Life Cycle Phase	Cost Elements Description
1	***Planning***	▪ Engineering effort cost based on the NCS solution design with respect to the mission, mission objectives, and mission requirements
2	***Acquisition***	▪ Cost of Acquiring the computer systems required based on NCS solution design
3	***Development***	▪ Cost of computer systems compliancy with the NCS architecture to include development, integration, testing, and deployment ▪ Cost of computer systems integration into the NCS architecture to include testing and deployment
4	***Operations and Support***	▪ Cost of managing, operating, sustaining, and supporting the NCS solution
5	***Decommission***	▪ Cost of de-installation of the NCS solution

Table 2. Cost Structure Element

able to calculate these decision attributes based on weighted priorities. The weighted priorities take into account the importance of each of the decision attributes and prioritizes each of the attributes based on historical information and experiences of the decision stakeholders. Therefore, during this section, a generic hierarchy or ranking process shall be considered in order to provide a solution that relies on the judgments of experts and subject matter experts to provide a priority or weighted factor on area of importance for each of the measuring attributes. For instance, if the mission requires a higher factor in mission reliability, then the process will take into account the importance of the reliability of the mission. This also goes along with the decision attribute of capability life cycle cost having a priority weight over the other decision attributes. In this case, if the life cycle cost requires a higher priority, subject matter experts weight it according to its importance. Further research is required in this area in order to determine the best approach in determining the feasibility of the NCS solution based on the decision attributes considered.

Way Forward

The work being performed in this area will provide a well-defined methodology in which a program office can utilize a decision process to determine the best feasible approach for satisfying an emerging capability. The approach hinges on the utilization of current operational or developmental systems to fulfill user requirements by taking advantage of existing systems. This paper defined a methodology framework to explore the selection of systems that can, when combined, provide a means to satisfy an emerging capability by minimizing the number of systems for development and utilizing current operational system capabilities that are fielded.

In future work as part of the selection process, the NCS solution will be verified and validated by attaining a measurable metric based on selected decision attributes that help determine the NCS effectiveness. The measurement for the NCS effectiveness will provide information to determine whether an investment in developing the NCS solution can be a viable commitment to successfully satisfy the operational requirement for the users. There is continued work to be performed in this area; however, this paper allows us to review a notional conceptual methodology in identifying decision attributes and using them as part of a process to identify an NCS solution for consideration. We will continually strive to identify and to quantify the preferred NCS solution to satisfy operational requirements. This paper will be followed by a detailed methodology, effectiveness models, and applications that will be applied toward an NCS solution to be considered and addressed.

There will be continuing work to be performed in this area to include a detailed methodology, effectiveness models, and application to an existing operational or notional mission. This will be a continued effort in the area of effectiveness measure in identifying and quantifying the preferred NCS solution in satisfying an operational requirement.

REFERENCES

1. Chelson, P. O. & Eckstein, R. E. (1971.) "Reliability Computation from Reliability Block Diagrams." Jet Propulsion Laboratory. California Institute of Technology, Pasadena, California.
2. National Research Council. (2015.) "Reliability Growth: Enhancing Defense System Reliability." Washington, D.C. National Academy Press.
3. DAU. (2016). "Measure of Effectiveness (MOE)." https://acc.dau.mil/CommunityBrowser.aspx?id=348978
4. Department of Defense. (2013.) "Defense acquisition guidebook." In Defense Acquisition Guidebook, Ed., 2013.
5. Department of Defense. (1980.) "Reliability Program for Systems and Equipment Development and Production." Vol. MIL-STD-785B. D. o. Defense, Ed., ed. Washington, D.C.
6. Office of Secretary Defense. (2014.) "Operating and Support Cost-Estimation Guide." Office of Secretary of Defense, Ed., ed. Washington, D.C.: Cost Assessment and Program Evaluation (CAPE).
7. Friedman, M. A.; Tran, P. Y., & Goddard, P. l. (1995.) "Hardware/Software System Reliability Modeling." in Reliability of Software Intensive Systems (Advanced Computing and Telecommunications Series). 1st Edition ed: William Andrew.
8. Jaynes, R. A. S. C.; Simpson, T.; Mallicoat, D.; Francisco, J.; Mizell, W. & Cikovic, D. (2012.) "Managing O&S Costs - A Framework to Consider."
9. Mccallam, D. (2013.) "Improving Enterprise Security through Cybersecurity Architecture Views."
10. Office of the Secretary of Defense, C. (2011.) "DoD Financial Management Policy and Procedures DoD 7000.14-R." D. o. D. F. M. Regulation, Ed., ed. Washington D.C.: Office of the Under Secretary of Defense (Comptroller).
11. Office, U. S. G. A. (2009.) "GAO Cost Estimating and Assessment Guide." GAO-09-3SP ed: GAO.
12. Sproles, N. (2001.) "Establishing Measures of Effectiveness for Command and Control: A Systems Engineering Perspective." 30.
13. Under Secretary of Defense for Acquisition, T. a. L., or USD(AT&L). (2003.) "The Defense Acquisition System Directive 5000.01." Vol. 5000.01. U. A. L. Department of Defense, Ed., DoD Directive 5000.1 ed. Washinton, D.C.
14. Under Secretary of Defense for Acquisition, T. a. L., or USD(AT&L). (2015.) "Operation of the Defense Acquisition System 5000.02." Vol. 5000.02. U. A. L. Department of Defense, Ed., ed. Washington, D.C.

ABOUT THE AUTHORS

Glenn Tolentino is a senior systems engineer for the Command and Control Department at Space and Naval Warfare Systems Center Pacific located in San Diego, Calif. During the past 23 years, Tolentino has been directly involved as a software and systems engineer in the design, development, integration and deployment of national level systems in the area of Command, Control, Computers, Communication, and Intelligence. Glenn's research interests include Systems of Systems, Mission Reliability, Capability Sustainment, and Systems Effectiveness. He earned a B.S. degree in applied mathematics from San Diego State University and an M.S. degree in software engineering from Southern Methodist University (SMU).
glenn.tolentino@navy.mil

Jeff Tian received B.S., M.S., and Ph.D. degrees from Xi'an Jiaotong University, Xi'an, China; Harvard University, Cambridge, Mass., USA; and the University of Maryland, College Park, Md., USA; respectively. He was with the IBM Toronto Lab from 1992 to 1995. Since 1995, he has been with Southern Methodist University, Dallas, Texas, USA, where he is currently a professor of computer science and engineering. He has been the associate director of the National Science Foundation Industry/University Cooperative Research Center for Net-Centric and Cloud Software and Systems (NSF NCSS IUCRC) since it was founded in 2009. Since 2012, he has also been a Shaanxi 100 Professor with the School of Computer Science, Northwestern Polytechnical University, Xi'an. His current research interests include software quality, reliability, usability, testing, measurement, and Web/service/cloud computing. Dr. Tian is a member of the ACM.
tian@lyle.smu.edu

Jerrell Stracener is Professor of Practice and founding director of the Southern Methodist University (SMU) Systems Engineering Program. He teaches graduate-level courses in engineering probability and statistics, systems reliability and availability analysis, and integrated logistics support (ILS). He performs and directs systems engineering research and supervises Ph.D. student research. Prior to joining SMU full time in January 2000, Dr. Stracener was employed by LTV/Vought/Northrop Grumman where he conducted and directed systems engineering studies and analysis as well as reliability engineering activities. He was also ILS program manager on many of the nation's most advanced military aircraft. Dr. Stracener was co-founder and leader of the SAE Reliability, Maintainability and Supportability (RMS) Division (G-11) and is an SAE Fellow and an AIAA Associate Fellow. Jerrell served in the U.S. Navy and earned both Ph.D. and M.S. degrees in statistics from SMU and a B.S. in mathematics from Arlington State College (now the University of Texas at Arlington).
jerrells@lyle.smu.edu

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 the areas of emphasis we are looking for:

Software Release Management
September/October 2017 Issue
Submission Deadline: Apr 10, 2017

The Profession
November/December 2017 Issue
Submission Deadline: Jun 10, 2016

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>.

Process is Easy, Change is Hard

Paul Kimmerly, Double Play Process Diagnostics

Abstract. Process is all around us. We follow processes in just about everything we do, from getting dressed in the morning to loading the dishwasher after a meal to walking the dog. Processes provide essential structure in life and in the workplace. Organizations rely on processes to get work done. However, organizations are always looking for ways to improve processes. This issue of "Crosstalk" covers a number of process improvement methodologies that organizations can use. All of these methods have their merits, but they have also experienced both success and struggle over the years. Why is it that process improvement methods sound so simple and straightforward but usually prove difficult to implement? The answer is simple – improvement methods all involve something very difficult: change. Choosing the process methodology is the easy part; managing the change and the resistance that comes with it makes it difficult.

Organizations may encourage change, but people typically don't like it. At face value, change is disruptive. Before people will embrace change, they need to understand the reasons behind it and how the change will benefit them. Resistance to change is inevitable, but organizations can manage change to help make desired improvements easier to achieve. As if organizational change isn't hard enough, a new generation is entering the workforce with different attitudes and views. The introduction of a new generation with a different point of view affects the dynamics of change. This article will examine processes, process improvement methods and change. How an organization manages change will determine how successful any process improvement effort will be.

The Easy Part: Process and Process Methodologies

What is a process? Simply put, a process is a series of related steps one follows to achieve a desired result. That end result could be providing a service, producing a product, or reaching an objective. Every organization follows processes, even if they don't realize it. Processes tend to arise naturally in an organization. When the people in an organization first try to do something, they realize that certain things must be done to accomplish the desired end result. As time goes on, these things and the sequence in which they are performed become habit or routine. People accept that certain things need to be done every time. Hence, a process is born. It may not be formally written down. It may not be followed exactly the same way each time. But it's there, and people accept it as the way things are done.

After people follow a process for a while, the organization takes notice. The organization may decide that things could be done better for any of a number of reasons. Organizations may want to do things faster, cheaper or with better results. When that happens, they start to look outside the organization for ways to improve their processes. Suddenly, organizations face a number of methodologies to choose from, including Lean Six Sigma, Agile, International Standards Organization (ISO) standards, Total Quality Management (TQM), and the Capability Maturity Model Integration (CMMI), among others.

Which one should an organization choose? Lean sounds good. Who wants a fat, bloated process? Six Sigma sounds scientific. It has a Greek word in it, after all. The ancient Greeks were pretty smart. Agile sounds good too. Who wouldn't want to be quick and agile? Slow and plodding doesn't sound nearly as good. ISO is international. That must be good, right? TQM must mean that the organization is in control of something because it is "managing" total quality, not just partial quality, but "total quality." The CMMI sounds pretty academic. It may not sound as riveting or catchy as some of the others, but an organization should want to be a mature grown-up about things. Mature capability sounds pretty good too. After all, who wants to be immature and incapable? It can come with a maturity rating too. That's even better – ratings look great on marketing brochures and corporate reports.

Next comes research into what all of these methods involve. When the organization reads the testimonials and thinks they understand a little about some of the methods, in come the consultants. When management meets the consultants, the conversation goes something like this:

Management: "We want to improve our processes."
Consultant: "Great! My methodology and I can help."
Management: "Which processes should I improve?"
Consultant: "Well, your processes."
Management: "I know, but which ones?"
Consultant: "The ones that are most important to you."
Management: "Umm … Which ones should those be?"

From there, the conversation moves on to a general discussion of process and organizational goals and needs. At some point, the consultant describes his or her methodology and how it can help. Management decides which processes need focused effort. When agreement is reached on the methodology and target processes, most process improvement efforts follow the same basic steps:

–Document the current process.
–Identify the desired improvements.
–Involve the affected people.
–Apply the selected improvement methodology.
–Deal with resistance to change.

The Hard Part: Managing Organizational Change

Regardless of methodology, improving processes is pretty simple, easy and straightforward up until changes to the process are actually proposed or made. How organizations manage the changes – and resistance to them – determines whether or not changes are successful. If change is managed well, improvement happens and the organization moves forward. If change is not managed well, the organization often moves on to the next "improvement du jour" and the cycle starts over again.

Resistance comes in many forms. Often, the first resistance comes from the fact that the process is how things have always been done. People are comfortable with it and know what to expect. Change involves uncertainty. People do not like uncertainty.

The resulting resistance can come in many forms:

- Attacking the proposal.
- False acceptance of the proposal.
- Avoidance of the new process.
- Constantly asking questions and acting confused about the new process.
- Never having time to commit to the improvement process.
- Agreeing with everything right up until the time to implement that change, and then finding a variety of questions and concerns. [1]

It's easy for people to understand ideas like providing better service to customers, building a better product, understanding requirements before beginning work, planning the work so you know what to expect, and managing against a plan to evaluate progress. However, as soon they hear that they need to change the way they do things, process improvement becomes hard.

Resistance tests an organization's commitment to improvement and change. For improvement efforts to be successful, organizations must be able to clearly state the reasons for change in a way that is meaningful to the people in the organization. No matter how cool they sound, statements like, "We want to be a world-class provider of choice," mean absolutely nothing to the people in the organization. The improvement efforts must be shown to support the work of the organization. The organization's mission will always win over process improvement efforts, and it should. Getting the work done is paramount. However, the improvement efforts must support getting the work done, or the organization is pursuing the wrong improvements.

Successful improvement efforts need more than a statement of what needs to be improved and why. Improvement requires resources and a plan. Management cannot simply supply resources and step back, waiting to hear when things are better. Management must show commitment to the change. The improvement efforts must be supported and reinforced in order to be successful. Peer pressure can be a great tool for management to use. Reinforcement can come from rewarding those that implement the change and applying pressure on those that do not. Management cannot afford to be vague about their support for process improvements.

Resources for process improvements are critical to successful change. These resources include the people needed to support and manage the change. Supporting infrastructure also enables improvement and includes the following:

- A process improvement plan.
- Documented processes.
- Organizational standards for processes, products and work environments.
- Organizational repositories for processes, measurements and lessons learned.

In order for a change to be successful, the entire workforce must be engaged. Different approaches work for different people. For example, as organizations integrate a new generation into the workplace, communication becomes a greater concern. The new generation looks at the world differently. They are used to always being connected. To them, old communication methods are just that — old and slow. Organizations must use a variety of tools to get the word out and to reinforce the need for improvement and change.

As the new generation finds its place and value in the workforce, longtime employees need to feel that they are not losing their value. Change may be appealing to the new generation, but they will also appreciate the support provided by an established process infrastructure. Long-term employees may feel that change threatens them, but an organization should draw on these employees' knowledge to help establish an infrastructure for the new employees to follow. Both old and new employees need to feel ownership of the processes and the associated improvement efforts. The entire workforce must be able to see that the change is organized and necessary. If they see things as disorganized or unnecessary, resistance will increase.

Conclusion

Organizations will find that the most difficult aspect of process improvement is truly the change associated with it, regardless of the improvement methodology chosen. Organizations should consider more than one methodology to find a good fit. Different improvement methodologies complement one another and can be used in combination. Something like the CMMI can provide an overall structure by providing a menu of process areas to focus improvement efforts, while Lean Six Sigma and Agile can help to streamline processes and remove wasteful activities in those process areas.

Once an organization decides which approach it will follow for process improvement, the focus needs to change from process methodology to managing organizational change. Choosing the process methodology is the easy part, but managing the change and the resistance that comes with it is more difficult. Organizations must set relatable improvement goals, plan their improvement like a project, provide resources, involve the entire workforce, manage the improvement project, reinforce desired behavior, and celebrate success. Paying attention to those details will help make change easier.

NOTES

1. For a more complete look at resistance and how to deal with it, see "Flawless Consulting" by Peter Bloch, 3rd edition, c. March 2011, Pfeiffer and Company, San Diego, Calif.

ABOUT THE AUTHOR

Paul Kimmerly worked for 25 years for the different incarnations of the United States Marine Corps Technology Services Organization (USMC TSO). He spent the last 16 years as the SEPG Lead before he retired in July 2011. Kimmerly is a certified CMMI High Maturity Lead Appraiser and instructor for the CMMI for Development and the CMMI for Acquisition. He works as an independent contractor with the CMMI Institute teaching and observing candidate lead appraisers and instructors. He is also a member of the editorial board for "Crosstalk" magazine. He contributed several articles on process improvement to "Crosstalk." The articles cover topics including organizational change, management's role in process improvement, acquisition, and high maturity concepts. Since retiring from government service, Kimmerly continues to work with clients in both the government and private industry as part of Double Play Process Diagnostics, Inc.

Double Play Process Diagnostics, Inc.
P.O. Box 17015
Pensacola, FL 32522
Phone: (913) 220-4499
Paul.kimmerly@doubleplayconsulting.com

In Search of a Modern Software Life Cycle

Secure DevOps Foundations for Large-Scale Software Systems

Don O'Neill

Abstract. "In Search of a Modern Software Life Cycle" explores the "Secure DevOps Foundations for Large Scale Software Systems" in terms of voices from the trenches, the field of play, life cycle on center stage, and evolutionary features and issues including sequential, prototype, incremental, iterative, spiral, CMMI©, technical debt, code and upload and frequency of release, next generation software engineering, open source software, false claims, integration engineering, and a new way of thinking.

Heard from the Trenches

If DevOps is needed to change the world, Secure DevOps is also needed to save the world. In a world where business questions masquerade as technical questions, where programmers must experience an epiphany before they are motivated to master the skill of writing secure code, [1] and where bonuses must be withheld to obtain management attention to security, resistance rules.

If these are the risks, what are the outcomes? Acquirers complain they don't know how to ask for secure code from vendors, adding that they get what they ask for but not what they want. [2] It's complicated! Programmers confess that writing code is hard, and writing secure code may be beyond the tipping point. [3] Software engineers wonder if there is any secure code anywhere and assert that best practices are insufficient. Supply Chain Risk Management Software Assurance practitioners retreat behind the wall and only hope for bug-free, patchable software deliveries accompanied by a bill of material. [4] These were just some of the comments made at the 2016 CERT Secure Coding Symposium conducted by the Software Engineering Institute in Arlington, Virginia, on September 8, 2016.

The Field of Play

Formed to support the advancement of software engineering in the Department of Defense (DoD), the Software Engineering Institute (SEI) lost its way by too vigorously pursuing commercial partners. Like the dog that chased and caught the firetruck without a plan for what comes next, the SEI lost its DoD sponsor, its principal foundation of financial support.

The impact of this lost sponsorship was most keenly felt by the Capability Maturity Model Integration (CMMI©) program, once the crown jewel of the SEI and Carnegie Mellon University (CMU) itself. Forced to depart the protection of the SEI and CMU, the CMMI© has now landed at the Information Systems Audit and Control Association (ISACA) in the form of the CMMI© Institute, relegated to serving the commercial IT governance professionals it catered to. Finding itself now in the competitive death grip of a more innovative and popular Agile method, the CMMI© framework continues to teeter. All this is occurring despite the fact that the value of the CMMI© has not yet been fully discovered (CrossTalk, 2012) despite a quarter-century of use. Yet there may still exist a way forward in harmonizing Agile and CMMI© (CrossTalk, 2016) as part of that discovery.

Even beyond the CMMI©, the broader software situation is dire (Defense AT&L, 2015). Industry and government continue to increase dependence on software produced by an immature profession that has stumbled in delivering trustworthy software components, systems, and systems of systems to the nation's critical infrastructure and defense industrial base. The result is cybersecurity weaknesses and vulnerabilities exploited at will by persistent adversaries whose capabilities and motivation can only be surmised by assessing their consequences.

Center Stage

At play on center stage in all this is the software development life cycle. Beginning with Winston Royce, managing the development of large software systems became the center of attention

based on a waterfall model of software activities and his belated inclusion of prototyping as an essential step (Royce, 1970).

From Royce's waterfall life cycle model followed by incremental, iterative, and spiral to the SEI's CMMI© followed by Agile methods and now DevOps, the software development life cycle continues as an unsettled issue. Today's unbridled complexity (Sheard, 2015), the stresses of scale in the Internet of Things (IoT) (Recode, 2016) with its explosion of endpoints and no one in charge, and the unpredictability of cybersecurity threats (CrossTalk, 2011) with their persistence of vulnerabilities like System 7 and its public safety access points all combine to destabilize software system development life cycle approaches.

At any point in time, Secure DevOps processes must possess the capability to detect cyber vulnerabilities and malware. Common Weaknesses Evaluation (CWE) and Common Vulnerabilities Evaluation (CVE) assist in this, as do tools like Hyperion from Oak Ridge, Function Extraction (FX) from CMU, MUSE from the Defense Advanced Research Projects Agency (DARPA), and Approximate Matching from the National Institute of Standards and Technology (NIST). Beyond the range of typical Secure DevOps, the sectors of the critical infrastructure with their stovepipe yet interdependent operations face more insidious supply chain resilience challenges (CrossTalk, 2014). And then there are cascade triggers. Hidden or in plain sight, cascade triggers are capable of invading various industry sectors in a variety of ways:

- The transportation sector can be brought to its knees if truck drivers cannot use credit cards to charge for gas tank fill-ups.
- The medical sector depends on the Internet to distribute and present patient electronic medical records.
- The electrical grid depends on a survivable electrical grid with predictable demand profiles matched to planned resources and capacities (Koppel, 2015).
- The banking and finance sector remains ever conscious of its need to protect next-day opening, even in the presence of a flash crash disruption (Lewis, 2014).
- The users of the telecommunications sector are increasingly vulnerable to Internet disruptions like Distributed Denial of Service (DDoS) and encryption-based scams like ransomware.

Evolutionary Features and Issues

The following life cycle evolutionary features and consequences are introduced, including sequential, prototype, incremental, iterative, spiral, CMMI©, technical debt, code and upload and frequency of release, next generation software engineering, open source software, false claims, integration engineering, and a new way of thinking.

Sequential

The much-aligned waterfall model is a linear sequence of dependent activities. Much of the focus on life cycle model improvement is devoted to disrupting this dependence on the sequential.

Prototype

The use of prototypes — perhaps rapid prototypes — is an attempt to produce an early kernel of operational capability that can be exercised (not so much tested) to glean necessary insights into selective component interactions, numerical analysis of algorithms and their finite word effects, computer capacity utilization of both memory and speed, and targeted operational usage considerations.

Incremental

The use of multi-level design (Defense AT&L, 2012) and staged incremental development (SSJ, 1983) are tactics to put early performance pressure on the development team and its people, processes, and tools through incremental stages of production; for example, operating system services, middleware, and environment; executing system and subsystem interfaces using underlying stubs; executing prime mode functionality buildup in place of stubs; and exercising and transitioning degraded mode scenarios.

Iterative

Larman skillfully traces the real-world application of various evolutionary features in his "Agile & Iterative Development: A Manager's Guide" (Larman, 2004). Larman mentions the work of the IBM Federal Systems Division (FSD) on the integration engineering of the Trident Submarine Command and Control System (SSJ, 1983) and its pioneering work on design, development, and management life cycle activities spanning advanced design, systematic design, systematic programming, code management, integration engineering, technical reviews, cost management, and program management (IBM SJ, 1980).

Spiral

Introduced by Barry Boehm, the foundational spiral method is a purposeful and strategic departure from the sequential waterfall model in integrating prototype, incremental, and iterative tactics in the systematic management of software system risk (Boehm, 2015).

CMMI©

Now that the CMMI© has been organized into three constellations for assuring an organization's capability to perform development, acquisition, and service, there is a need to extend the range of value of the CMMI© to a new normal (CrossTalk, 2012). As an organization improves its process maturity, strategic imperatives need to replace waste and neglect as the CMMI© value driver. Only those organizations able to elevate their game and transition from tactical to strategic use of the CMMI© will be able to reap its full value.

While the traditional treatment of the value of the CMMI© in terms of cost, schedule, productivity, quality, customer satisfaction, and return on investment is sufficient to promote adoption of the CMMI© and even to sustain a process improvement initiative through the early maturity levels, the value of the CMMI© determined in this way is likely to be underestimated as the organization approaches higher maturity levels.

The value of the CMMI© can be framed more strategically as a means for carrying out visionary statements of strategic intent in achieving measured outcomes in business and competitiveness, management and predictability, process and improvement, engineering and trustworthiness, and operations and dependability.

Technical Debt

Technical debt is the organizational, project, or engineering neglect of known good practices that can result in persistent public, user, customer, staff, reputation, or financial cost (Defense AT&L, 2013). Shortcuts, expedient activities, and poor practices that contribute to the initial product launch or initial operational capability are often cited as justifiable excuses for taking on technical debt. But in truth, most technical debt is taken on without this strategic intent, without even knowing it, and without the capacity to do the job right.

Code and Upload and Frequency of Release

In order to simplify, relieve stress and sustain a very high frequency of release, one major corporation is employing an extreme move. They no longer test software upgrades, preferring instead to use the code-and-upload tactic. This leaves any defects to be encountered by unsuspecting customers. The frequency of release cited by this corporation is an amazing 30,000 per year.

Next Generation Software Engineering

Practical Next Generation Software Engineering addresses the unclaimed benefits and unmet needs associated with competitiveness, security and software. In accordance with the austerity of the times, the immediate goal of practical next generation software engineering is to drive systems and software engineering to do "more with less … fast" (IEEE, 2009). Four practical objectives are identified to advance this goal using smart, trusted technologies:

- Drive user domain awareness.
- Simplify and produce systems and software using a shortened development life cycle.
- Compose and field trustworthy applications and systems from parts.
- Compose and operate resilient systems of systems from systems.

Open Source Software

Open Source Software is openly available off-the-shelf software that depends on community development and distribution support subject to license compliance. Open source code is openly available for inspection and change. By contrast, closed source is a proprietary product dependent on the vendor for support and not open to inspection or modification.

Open source software features free distribution of source code. When open source software is extended or revised, the result is termed a "derived work." Furthermore, an open source software license may permit resale of a derived work. While freely available, there are project costs associated with modifying and integrating derived works into deployable software systems.

The proper use, reuse, modification, and sale of open source software as derived work lies in the art of program and contract management. When this is done in government contracting, retaining the classification of "Commercial Off the Shelf" (COTS) and "Government Off the Shelf" (GOTS) software has financial and legal consequences. Furthermore, blending all this into use under the General Services Administration (GSA) contract may introduce complexities not yet fully explored. The government is recognizing the potential savings in absorbing software into the GOTS classification and is now establishing target goals for accomplishing this. Failure to assign the proper COTS and GOTS classifications and associated fee structures may result in a Department of Justice (DOJ) false claims charge against the contractor under the False Claims Act.

False Claims

With 80 percent of government software procured as COTS and accorded limited or restricted rights, government acquisition managers need to be aware of intellectual property considerations (Defense AT&L, 2014). When modified and extended through government funding, COTS software becomes GOTS software and is entitled to government purpose rights. Unless the government acquisition manager insists on it, a contractor may engage in false claims practice by improperly marketing and selling GOTS software products as COTS. Instead of receiving the benefits of government purpose rights, the government may be charged a commercial product licensing fee and accorded only limited or restricted rights. Neglecting intellectual property rights can be costly.

Integration Engineering

The penultimate challenge in fielding large-scale systems and systems of systems that are trustworthy, secure and resilient resides in critical infrastructure (White House, 2016). Simply put, the resilience value proposition is intended to yield a critical infrastructure capable of anticipating, avoiding, withstanding, minimizing, and recovering from the effects of adversity, whether natural or man-made, under all circumstances. This is based on an architecture of resilience that squarely faces the issues of harmonizing a diverse industry sector culture and context and offers effective prescriptions for success in the form of well-trained intelligent middlemen, a resiliency maturity framework, a system of systems technical architecture, a common and useful way of working, and an integration engineering program structure staffed by a capable resilience integrator. Anticipation and avoidance replace cleanup, recovery, and opportunity loss.

The author offers the following integration engineering context and culture harmonization guidance:

- Formality within an architectural framework facilitates the imposition of distributed supervisory control, interoperability, and operation sensing and monitoring protocols.
- Strong code management practices facilitate reconfiguration and reconstitution.
- Exercising strong control over the workforce facilitates business continuity and survivability.
- Exercising strong government control facilitates compliance for the benefit of the commons at the expense of initiative for the self-interest.
- The diverse industry sector expectations of trust, loyalty, and satisfaction must be respected, blended, and harmonized.
- Technical debt must be eliminated.
- Cascading and propagating triggers must be anticipated,

avoided, and minimized.
- Industry sector software sourcing exposures must be understood and managed.
- Supply chain risk management operations must be assured.
- Cybersecurity strategy policy decisions and defined tactics must be assured.

A New Way of Thinking

The Integration Engineers, Resilience Integrators, and Intelligent Middlemen must be equipped with a new way of thinking. (Jacobson, I., Lawson, H.B., 2015). As the twig is bent, so grows the tree. To get your project off on the right foot, expectations should be set and evidence should be sought on the following assertions and principles:

- Stakeholders are in agreement and share a vision for the project.
- An opportunity value proposition has been established, and stakeholders share a vision for achieving it.
- Requirements or user stories are coherent and acceptable, and stakeholders share a vision for them.
- The software system architecture is selected and comprises a domain-specific architecture to guide software system implementation. The software system implementation is also made ready and operational with no technical debt.
- The team operates in collaboration, shares a vision for the project, and is ready to perform with respect to shared vision, software engineering processes, software project management, software product engineering, operations support, and domain-specific architecture processes, methods, and tools.
- The way of working by the team has established foundations for software engineering processes, software project management, software product engineering, and operations support.
- Work begins only when everything is prepared, including coherent requirements and acceptable user stories, stakeholders that are in agreement, and an established foundation for the way of working.
- All work products are prepared and inspected in accordance with a defined standard of excellence assuring completeness, correctness and consistency.

A product focus on perfection is assisted by the "work product" expectations as shown here. The work product should be:

- Identified as part of the way of working.
- Produced, shared with the team, and inspected.
- Complete with parts that are traceable to predecessor work products.
- Correct with parts that are verified and provably correct.
- Consistent in style and form of recording, and consistent with the software system architecture and its rules of construction.
- "Value add," traceable to user stories and the "Done" criteria for the way of working.

Conclusion

Clearly the search for an ideal model software life cycle is a journey, not a destination. The disruptive journey continues, with the tension of Agile and cybersecurity serving as current disrupters. As before, a variety of adaptations and innovations will emerge from practice, and some will be absorbed in the body of professional practice for those that follow. And so goes the evolution of the software profession.

ABOUT THE AUTHOR

Don O'Neill served as the president of the Center for National Software Studies (CNSS) from 2005 to 2008. Following 27 years with IBM's Federal Systems Division (FSD), he completed a three-year residency at Carnegie Mellon University's Software Engineering Institute (SEI) under IBM's Technical Academic Career Program and has served as an SEI visiting scientist. A seasoned software engineering manager, technologist, independent consultant, and expert witness, he has a Bachelor of Science degree in mathematics from Dickinson College in Carlisle, Penn. His current research is directed at public policy strategies for deploying resiliency in the nation's critical infrastructure; disruptive game-changing fixed price contracting tactics to achieve DOD austerity; smart and trusted tactics and practices in supply chain risk management assurance; a defined "software clean room method" for transforming a proprietary system into a clean system devoid of proprietary information, copyrighted material, and trade secrets and confirming, verifying, and validating the results; and a constructive approach to sequencing the transition of SEMAT Essence Kernel Alpha states with an eye to pinpointing the risk triggers that threaten success and lead to the accumulation of technical debt.

NOTES

1. David Svoboda. (Sept. 8, 2016.) SEI CERT Secure Coding Team, 2016 CERT Secure Coding Symposium. Arlington, Virginia.
2. Kris Britton. (Sept. 8, 2016.) NSA Center for Assured Software, 2016 CERT Secure Coding Symposium. Arlington, Virginia.
3. Dr. Carl Woody. (Sept. 8, 2016.) CERT Cyber Security Engineering Team, 2016 CERT Secure Coding Symposium. Arlington, Virginia.
4. Josh Corman. (Sept. 8, 2016.) The Atlantic Council, 2016 CERT Secure Coding Symposium. Arlington, Virginia.

REFERENCES

O'Neill, D. (January/February 2012.) "Extending the Value of the CMMI to a New Normal." CrossTalk, The Journal of Defense Software Engineering. http://www.crosstalkonline.org/storage/issue-archives/2012/201201/201201-ONeill.pdf.

O'Neill, D. (July/August 2016.) "The Way Forward: A Strategy for Harmonizing Agile and CMMI." CrossTalk, The Journal of Defense Software Engineering. http://static1.1.sqspcdn.com/static/f/702523/27124563/1466890559753/201607-ONeill.pdf?token=CYClitqj%2B4Q5KzD%2B8d1nHTuHh9s%3D.

REFERENCES CONT.

O'Neill, D. (May/June 2015.) "Software 2015: Situation Dire." Defense Advanced Technology and Logistics (DAT&L) Magazine. http://www.dau.mil/publications/DefenseATL/DATLFiles/May-Jun2015/O'Neill.pdf.

Royce, Winston W. (August 25–28, 1970.) "Managing the Development of Large Software Systems." Technical Papers of Western Electronic Show and Convention (WesCon). Los Angeles, Calif., USA.

Sheard, Sarah. (2015.) "Chapter 5: Complexity, Systems, and Software, 'Software Engineering in the Systems Context.'" Edited by Ivar Jacobson and Harold "Bud" Lawson. College Publications, King's College, London. ISBN 978-1-84890-76-6. 578 pages;

O'Neill, D. (1983.) "Integration Engineering Perspective." The Journal of Systems and Software, 3. 77-83. http://www.sciencedirect.com/science/article/pii/0164121283900067.

O'Donnell, Bob. (June 22, 2016.) "The Internet of Things is facing challenges with scale." Recode. http://www.recode.net/2016/6/22/11991414/internet-of-things-iot-challenges-scale.

O'Neill, D. (September/October 2011.) "Cyber Strategy, Analytics, and Tradeoffs: A Cyber Tactics Study." CrossTalk, The Journal of Defense Software Engineering. http://www.crosstalkonline.org/storage/issue-archives/2011/201109/201109-ONeill.pdf.

O'Neill, D. (March/April 2014.),"Software and Supply Chain Risk Management Assurance Framework." CrossTalk, The Journal of Defense Software Engineering. http://www.crosstalkonline.org/storage/issue-archives/2014/201403/201403-ONeill.pdf.

Koppel, T. (2015.) "Lights Out." Crown Publishing Group. ISBN 978-0-553-41996-2. 277 pages.

O'Neill, D. (1983.) "Integration Engineering Perspective." The Journal of Systems and Software, 3, 77-83. http://www.sciencedirect.com/science/article/pii/0164121283900067.

Larman, C. (2004.) "Agile & Iterative Development: A Manager's Guide." Pearson Education, Inc. ISBN 0-13-111155-8, 82-85.

O'Neill, D., Linger, R.C., Dyer, M. & Quinnan, R.E. (1980.) "The Management of Software Engineering." IBM Systems Journal, Vol. 19, Number 4, 414-477. http://www.research.ibm.com/journal/sj/.

Boehm, Barry. (2015.) "Chapter 6: Principles and Rationale for Successful Systems and Software Processes, 'Software Engineering in the Systems Context.'" Edited by Ivar Jacobson and Harold "Bud" Lawson. College Publications, King's College, London. ISBN 978-1-84890-76-6. 578 pages.

O'Neill, D. (March/April 2013.) "Technical Debt in the Code: Cost to Software Planning." Defense Advanced Technology and Logistics (DAT&L) Magazine. http://www.dau.mil/pubscats/ATL%20Docs/Mar_Apr_2013/O%27Neill.pdf.

O'Neill, D. (June 2009.) "Preparing the Ground for Next Generation Software Engineering." IEEE Reliability Society, Annual Technology Report 2008, 148-151.

O'Neill, D. (November/December 2014.) "Avoiding Proprietary Problems: A Software Clean-Room Method." Defense AT&L Magazine. http://www.dau.mil/publications/DefenseATL/DATLFiles/Nov-Dec2014/O'Neill.pdf.

O'Neill, D. (April 14, 2016.) "Integration Engineering in the Pursuit of Critical Infrastructure Resilience: A Unified Theory." White House Cyber Commission on Enhancing National Cybersecurity, Kickoff Meeting. http://www.nist.gov/cybercommission/upload/Meeting_Minutes_April_14.pdf.

Jacobson, I. & Lawson, H.B. (2015.) "Software Engineering in the Systems Context." Edited by Ivar Jacobson and Harold "Bud" Lawson. College Publications, King's College, London. ISBN 978-1-84890-76-6. 578 pages.

Lewis, Michael. (2014.) "Flash Boys: A Wall Street Revolt." W.W. Norton and Company, Ltd. ISBN 978-0-393-24466-3. 274 pages.

The Software Deployment Process and Automation

Dr. Nary Subramanian, Associate Professor of Computer Science, University of Texas at Tyler

Abstract. Software deployment is the last step in the software development life cycle. During deployment, control of the software transfers from the development team to the customer. After deployment, people in the customer organization will use the software as part of their jobs and derive economic benefits from the software. Any defects found in software post-deployment are resolved as part of the maintenance phase. The first step in mitigating user problems is the proper deployment of software. Software deployment is anything but trivial. Some enterprise software may take months, if not years, to completely deploy. Therefore, efficient software deployment will considerably shorten the deployment phase and save resources in terms of cost and labor. In this article, we explore typical models for software deployment. Based on these models, we develop a generic software deployment model, then identify deployment processes that lend themselves to further automation and may lead to an overall reduction in the deployment effort.

1. Introduction

Software deployment or installation represents the final handover of software from the development team to the customer. After successful deployment, the software system is finally operational so that the customer can benefit economically from its use. At the end of this deployment effort, the software development organization receives payment from the customer and the project is considered successful from both the developer's and the customer's viewpoints. However, software deployment is anything but trivial, depending on the scale of implementation. While a nontechnical person can install a desktop application by either installing a downloaded file or installing from a disk, a large-scale enterprise resource planning (ERP) system such as SAP may take several months — if not years — to be fully configured and ready to use [1, 2, 3].

A question one might have is why certain software deployments take a long time. Is it possible to shorten all deployments to the time it takes to install a desktop application? In this article we examine typical deployment models and discuss some answers to these questions. To answer these questions, we develop a generic deployment model based on typical deployment models, and this generic model will help us rationalize our answers. We also explore opportunities to automate some or all deployment activities.

What happens when, after successful software deployment, users notice defects (or bugs) during normal software operation? The customer reports these bugs to the software develop-

ment organization's help desk. From there, the software enters its maintenance phase. But a prerequisite for a normal operational state is a successful deployment effort from the software deployment team. In this article, "software developer" refers to the organization that developed the software, while "customer" refers to the organization that has procured the software and will deploy it.

What happens when the deployment effort fails? The failure may have been due to misunderstood system configuration requirements on the customer's end. This means that improper or insufficient hardware resources (including CPU, memory and network bandwidth) were allocated, or that required software (including databases, servers and operating systems) was not provisioned. If the customer is not willing to accept incomplete software — that is, software that has defects or does not satisfy requirements — then the software deployment may need to be scrapped, the developer may not be paid any pending invoices, or (in extreme cases) delayed or failed deployment may lead to litigation[4]. In all cases, failed deployment leads to increased cost for both the software developer and the customer and to unhappy users at the customer organization. An example of this is the FBI's Sentinel project that incurred extensive time and budget overruns and still did not satisfy its users; [5] also, it has been reported that only 7 percent of ERP projects are expected to be deployed on time. [6] Proper understanding of the deployment effort and its success are essential before the software is useful to the customer.

2. Software Deployment Models

Figure 1 depicts the typical model for software deployment for small-scale software, such as software for home use or for a small company. The first step in this deployment is to verify that software meets the user's requirements, including functional requirements like features as well as nonfunctional requirements like performance and reliability. For custom-developed software, this is typically done by demonstrations and references. For COTS (commercial off-the-shelf) software, this is usually done by running trial freeware versions.

At this stage the customer also verifies the constraints imposed on the software, such as the hardware requirements and the operating system. If the software satisfies all requirements, then the customer compares its cost and features to similar software from other vendors and decides on a specific software and vendor. The customer then purchases the software and either downloads it or inserts a disc containing the software in a computer. The software executable is then run either by double-clicking, by issuing a command on the terminal, or by using a GUI-driven setup program. During this process, any activation keys issued by the software manufacturer need to be entered, and the customer must agree to the end-user license agreement. The software is then installed, which is followed by the almost inevitable reboot or restart of the computer. Upon restarting, the software asks for configuration information such as language, time zones, user information and so on. If the software requires access to other systems, like email or the Internet, then this integration information is also provided. For example, when an app is installed on a smart phone, integration authorization is often required. After this, the software is ready to use.

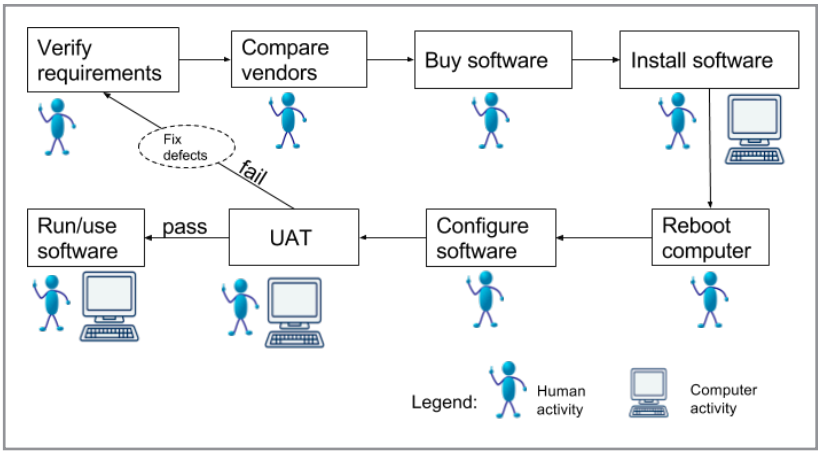


Figure 1. Deployment Process for Small-Scale Software

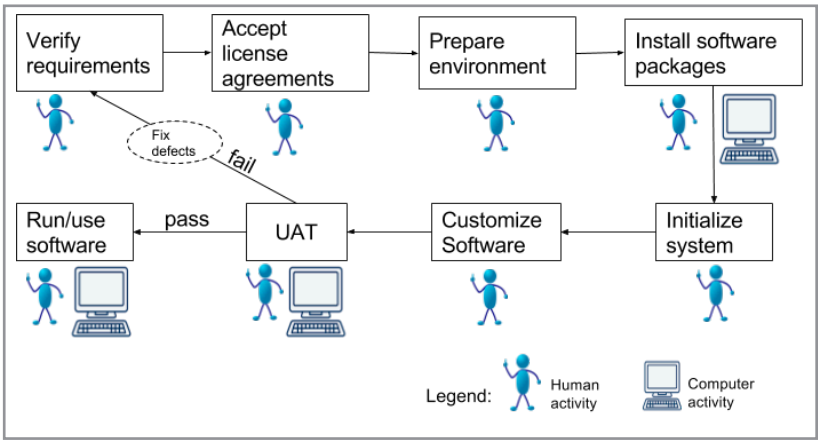


Figure 2. Deployment Process for an IBM Machine

However, the customer has to perform the user acceptance test (UAT) to ensure that the purchased software actually satisfies the requirements in the customer's computer. If UAT fails, the customer can contact the developer to fix defects or get a refund. However, if the UAT is successful, the customer can use the software.

Many of us have followed this deployment process when installing shrink-wrapped software on our computers for many years. Today, apps for smartphones are also installed by following most of the steps in Figure 1. As can be seen, the deployment process has several manual steps, and the computer-based activities are limited to the installation, testing, and execution of the software. Therefore, even though we tend to think of software installation as a one-click process, there is usually significant time spent waiting for the computer to complete executing the code based on our inputs. We also must perform several manual steps, which we tend not to include in the time it takes to install software for personal use or small office use. However, the whole process does not last more than an hour for most normal deployments.

Now let's consider the process for installing software on an IBM machine. [7] As mentioned in this reference, the whole process can take several days, if not weeks. The deployment process for the IBM machine may be abstracted, as shown in Figure 2.

For the IBM deployment process, the first step is to verify the hardware and system requirements for software. Typical system

requirements include the ability to restart the system, ability to log in and out, and the ability to install fixes. Subsequently, the deployment team obtains the hardware and software resources needed for software deployment. Any license agreements that need to be signed before the software is installed must be completed, and the server must be prepared for installation, including prepared to accept the media used for software distribution. The distribution media used for software is then attached to or inserted into the system, and the OS and any licensed code for running the software are first installed. The software package is then installed, and the entire system is initialized so that the software package installation is complete. Another aspect covered during initialization is any integration with other systems, including collaboration systems (email, calendar) or authentication (for single login) or payment gateways. Software is then configured as required and tested by users. Any data migration or addition of critical mass of data also happen at this step. If defects are found during UAT, the developer is required to fix them. The cycle then repeats with any changes during defect-fixing incorporated in the process. If the UAT passes, then the software can be used normally. In the IBM deployment process and small-scale deployment, the installation, UAT, and normal use are automated to a large extent. Remaining steps require substantial human intervention.

Figure 3 shows the deployment process for a cloud-based system. For example, documentation for OpenStack, an open-source cloud platform, deployment may be seen in [8]. The typical deployment process starts again with understanding the hardware, software, and networking requirements for the software. These include considerations of average and peak CPU, RAM, disk, and network input and output requirements. To get a clear picture of these requirements, the deployment team needs input from the sales and marketing teams as well. For example, the sales team can provide the total number of expected users, while the marketing team can predict peak loads to expect during sales promotion activities. Using this information, the deployment team can compare the offerings of different cloud service providers. For example, the rates and standard flavors supplied by different cloud vendors need to be matched with the requirements. It is possible that none of the offerings are good enough. If so, the deployment team will have to install the software on a bare-metal server. Once the vendor and solution have been decided, the system will need to be procured and appropriate service-level agreements signed. Subsequently, either the provider or the deployment team will need to spin up the virtual machines, networks, and external gateways based on system requirements. After that, the software packages required are installed in the VMs. The system is then initialized, which includes connecting the application with its front and back ends. Again, during initialization, integration with any external systems (such as email servers, authentication servers or payment gateways) are also performed. Then comes the configuration phase, when the software is customized to be able to add users or customers. Also during configuration, any licenses for software to be used are also activated. UAT is then done on the software, and the developer fixes any defects that arise. The cycle then repeats from the requirements phase with the modified software. Finally, upon successful UAT, the software enters the production phase. In the cloud deployment, the installation, UAT, and normal use are also automated; the remaining steps are largely manual. Typical cloud deployment can take anywhere from a few hours to a few weeks [9] depending on the manual parts of the process and defect fixing.

3. The Generic Deployment Model

After analyzing the typical processes for deploying systems at three different scales, we can create a generic model for software deployment and discuss the reasons for frequent unsatisfactory deployments. We can also recommend ways to improve the situation. The generic model is shown in Figure 4.

The generic model has eight steps. The first step is the "verification" process, when the software requirements are established in terms of the hardware and software required for deploying the software. These include networking requirements, CPU, RAM, disk, backups, recovery processes, security appliances, and so on. In addition, a developer must identify the operating system, databases, servers, and other software requirements at this point. The next step in the deployment process is the "negotiation" phase, when the deployment team negotiates the best offer for both hardware and software from vendors or the IT team (if in-house). Any service-level agreements are also negotiated at this time. The third step is to "procure" the best possible solution. This may include ordering the items and getting them shipped and delivered for on-premise deployments or getting the appropriate third-party provider to set up the solution for hosted or cloud deployments. The fourth step is "installation," when the software environment (including the operating system, databases, servers, and the like) are installed on hardware or appropriate virtual machines spun off for cloud deployments. All required software packages are installed once the environment is installed. The fifth step is "initialization," when the software and hardware systems are started up and global settings for administration credentials, licenses, database schema, and the like are established. Any external network access for emails, the payment gateway or the Internet are also established. The sixth step is the "configuration" step, when the number of users, their access credentials, their authorizations, their memory restrictions, and so on are configured in the software. Any data migration from a legacy system, adding of sufficient data for users, and vendor licenses are also established in this step. The seventh step is the "User Acceptance Testing (UAT)," when users test the software to ensure it satisfies its requirements including performance, security and reliability. If any defects are found during this step, then it is sent to the developer for fixing and the cycle restarts with the modified software. In the eighth and final step, the UAT is passed, the software is used for production, and economic benefits are derived.

As shown in Figure 4, most of the steps in the deployment model are manual. The only steps that are mostly automated are the installation, UAT, and use of the software. All the remaining steps require extensive human involvement. The step "Fixing Defects" is usually the responsibility of the developer, and therefore is outside the scope of the deployment organization. However, if defects are due to improper deployment in the steps so far, then

the deployment organization must step in to fix them.

Table 1 gives the activities performed during each step of the generic deployment model and approximate timeframes. Each of the steps in this table is discussed in section 4.

3.1 Applying the Generic Deployment Model to Common Software Deployment

In this subsection we will apply the generic deployment model to the deployment of three common types of software: web applications, SaaS, and mobile applications. This discussion will allow us to understand the extent of this model's applicability.

A. Deploying Web Applications

There are several steps involved in the deployment of web applications[10]. Three layers characterize a typical web application: the "database layer" or back end, the "application layer" or business logic, and the web server that serves the web pages. Therefore, all three layers need to be understood during the verification step, including the product's brand name, its operating system, and the hardware's sizing requirements. During the negotiation phase, the software and hardware required for deployment are procured from either internal resources or external vendors, and any agreements for this procurement are signed. During the procurement phase, the software and hardware are obtained. For open source software such as LAMP (Linux, Apache, MariaDB or MySQL, and PHP) the software is downloaded into the hardware and installed. The operating system, database software, application software, and web server are first installed during installation phase, then the web application is also installed. During initialization, the empty database tables are created using scripts, super user is created, initial users for the system are registered, and the application is set to a known initial state. During configuration, access rights for users are set, authorizations are configured, any needed database migration is done, and any needed connections to payment gateways are established. During UAT, a small initial set of users test the system. If no defects are found, the system is open to all users. If defects are found, the process repeats from the stage where the defect originated.

B. Deploying SaaS Applications

The first step when deploying a SaaS application is to verify its requirements — the operating system, the back end, the front end, the hardware requirements, backup and disaster recovery procedures and so on. Then negotiation takes place with different cloud service vendors who can provide the software and hardware to satisfy the requirements and the service level agreements. In fact, there is usually a checklist [11] that must be completed before SaaS can be deployed. Negotiation is also required if the software has to interface with third-party application providers or if any agreements need to be signed. During the procurement phase, the required virtual environment for hosting the SaaS is deployed. The software is installed in the virtual environment during the installation phase. SaaS is then initialized with the data necessary for its operation, including the pointers to the databases, networks, and backup devices. During configuration users are created, the access rights matrix is established, user constraints on memory and processor usage are

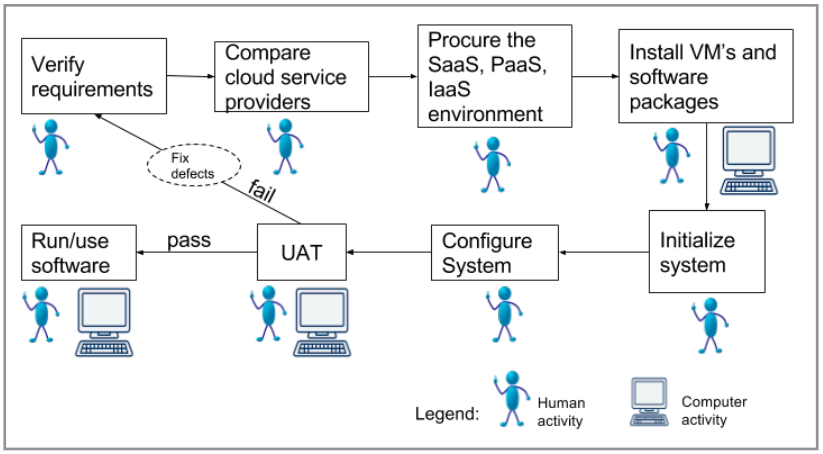


Figure 3. Deployment Process for Software in the Cloud

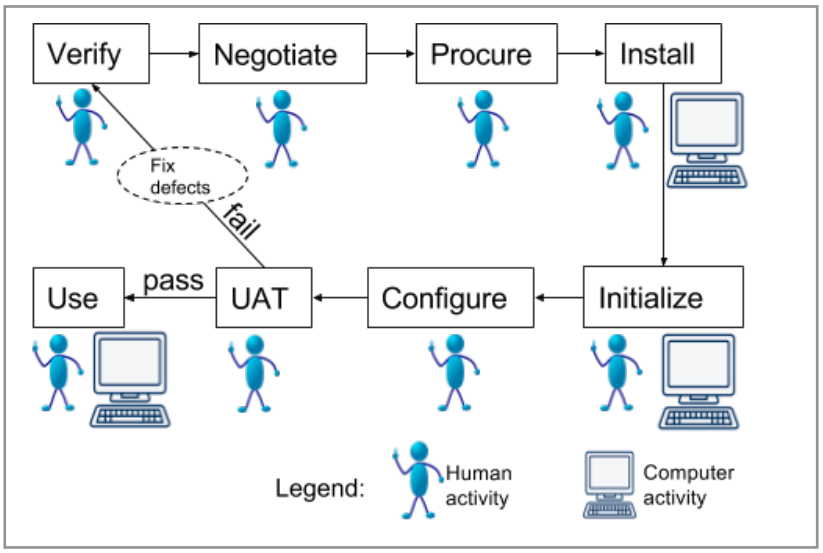


Figure 4. The Generic Deployment Model

established, payment gateway interconnections are established, and connections with other external APIs (Application Programming Interfaces) are also established. During UAT, the initial set of users test the system and if no defects are found, the system is open to all users. If defects are found, then the process is repeated from the stage where the defect originated.

C. Deploying Mobile Applications

When we think of deploying a mobile app, we usually think of downloading and using it. However, even a trivial app must go through several steps. We first verify the need for the app, even if the need is simple curiosity. We then decide where to download it from — the official store for the device (AppStore for iOS devices and Google Play Store for Android) or a third party store (such as Amazon.com). We then purchase the software if it is not free. After that, we download the app during the install phase. During initialization, we give permissions for the app to access directories, devices, and logs. During the configuration phase, if needed, we set up permissions to use the device, location settings, the look and feel of the user interface, and so on. UAT is the actual trial run of the software to see if it works

as we expected. If we like the app we keep it; otherwise, we uninstall or disable it. Optionally, we may choose to report bugs to the vendor or, if we are happy, give the vendor a good review.

However, for an enterprise app, there are more stringent procedures at each step since such apps are usually frontends for a web application, a server application, or a SaaS application. These apps are approved for download by the enterprise IT team. Therefore, there are two flavors — those installed by the enterprise IT on enterprise devices and those installed by the employee on his or her device following a BYOD (Bring Your Own Device) policy. In either case, the steps are mostly similar. First, verify that the app is indeed the frontend for the enterprise application and will perform most of the functions on the mobile device. Next, negotiate with the vendor on the price or compatibility with different platforms and sign any agreements for support and upgrades. Third, procure the software if it is distributed by an enterprise device manager [12] or else download the app from the appropriate app store during the install phase. During initialization, give permissions for the app to access personal information on the device and sign any end-user agreements. During configuration, enter the authentication credentials into the device as well as the path to the backend systems and the frequency of synchronization of data with the backend. During UAT, use the device and see if it meets the established expectations. If it does, we continue to use it. Otherwise, we report issues to the IT (if downloaded by the employee) or to the vendor (if downloaded by the IT).

For enterprise apps there are also associated issues of mobile device management, mobile device enrollment, and configuring mobile device settings. Device management [12] distributes applications, configuration settings, and security tools (for example, some organizations encrypt devices for using their apps). Device management can be part of the negotiation, installation, initialization, and configuration steps. It usually becomes a big part of normal mobile device operation after deployment in case devices get lost or become part of a botnet (the device manager can either locate the device or remotely wipe it). Mobile device enrollment [13] allows the organization to enroll many devices at the same time and to monitor and manage their use. Enrollment can be part of the verification, initialization, and configuration steps. Mobile device configuration [14] manages mobile device settings and is usually part of the configuration step.

As can be seen, we are not focusing on deploying a mobile app to the app store, [15] which is part of the app development process.

4. Analyzing the Performance of the Generic Deployment Model

As can be seen from Table 1, software deployment takes a long time, even when automated tools are used. This is mainly because of the extensive human involvement in the process. Even though actual software usage occurs over years, it can take several days or months before the benefits from software usage can begin to accrue. We will now analyze each step and see if further automation of the step can help reduce deployment time.

In the first step, the deployment team needs to understand software requirements and obtain sizing estimates. The deployment team often requires a detailed specification sheet so that any deviations from the organization's IT procurement policies may be identified early and budgeted for. Organization compliance teams, including audit and risk management divisions, may also be involved to identify any risks to the organization's information security that may be caused by the deployment of the software. Other professionals, like consultants (who may help with the integration effort later), lawyers (who may help clarify the requirements terms), networking specialists, database experts, and others may also be involved. Thus, the typical timeframe can be anywhere from a few days to months. However, for smartphone apps and small-scale software for home or small business use, a team of professionals is rarely used even though there are reports that privacy policies for some apps may invite legal attention[16].

However, there is very little scope for automating this effort, even if requirements are written using a standard such as BPEL (Business Process Execution Language) [17] because cross-domain human expertise may be required for this step.

In the second step, "negotiation," the agreements are signed with the software developer and the negotiation of cost and delivery with hardware and support software vendors takes place. Therefore, several divisions, including legal, procurement, and finance, could be involved in this process. The legal team is involved to ensure the customer organization is using standard clauses in its agreements and no unreasonable terms are involved. Purchasing is involved to make sure contracted standard vendors are used and any deviations are approved. Finance is needed for payment. Again, the cross-domain expertise required for this step makes automation difficult. Therefore, time needed for this step can be anywhere from a few days to months, especially if the software will be used in several countries and legal ramifications of multi-country use need to be explored. Software for negotiation is still in the research stage [18] and does not seem to have been adopted in mainstream legal practice, [19] so there is little scope for automation in this step.

The third step is procurement, when IT infrastructure is ordered for deployment. This ordering process can be largely automated using an ERP like SAP, [20] and software can be downloaded. However, hardware has to be physically delivered, and this takes time — especially if specialized made-to-order hardware is involved in deployment. Therefore, this step is mostly manual.

The fourth step is installation of software, and this step is usually automatic. However, strict checklists used by organizations during this process and any consultant input will add time to this step. Sometimes corporate audit and review teams need to certify that the installation was successful; this can also delay this step.

The fifth step is the initialization step and involves several manual activities. During initialization disk partitions are created; accesses to network, databases, network attached storages (NAS), and authentication and authorization systems (for example, active directory integration) are provided; and user keys for encryption are provided. Integration with external systems, such as emails, payment gateways, and the Internet, also occur during this step. The company account will need to be used for integrating with such external systems for proper financial

accounting purposes. Use of initialization scripts can automate parts of this process, but subsequent validation of all initialization steps is usually done manually.

In the sixth step, the configuration step, the software is configured for use. This includes assigning users to the software, updating database schemas, and any other configuration needed for proper use of the software. Data migration also takes place at this point so the system is ready for use, especially if the system is replacing a legacy system. For new systems, critical mass of data need to be entered into the system before it can be used. For example, in a Document Management System, [21] scans of existing physical documents need to be input into the system before the system becomes useful. Any vendor licenses required for using the software are also activated during this step. Backup procedures are also implemented at this stage so that disaster recovery procedures are ready once the system becomes operational. This configuration step is usually performed manually, though several automated configuration management tools [22] that run configuration scripts are now appearing in the market to automate parts of this process.

The seventh step is UAT. During this step, the software is tested to ensure that it is ready for use. Users are provided software training, then use the software as expected. Any defects found during this step are fixed. These defects may be in the software, which means the software developer fixes them. Or, if the defects are due to improper implementation of the deployment steps, the deployment organization fixes them. Once the software or the deployment activities are corrected, the UAT continues until users consider the software acceptable. Several automated tools exist for testing, [23] but this step is still largely manual [24] and can take several weeks to complete.

The final step of the deployment process is "going live" with the new software, which means the software is put to normal use. There are several issues to consider during this step, including the cut-over process, the establishment of a help desk for maintenance, a provision for regular user training, proper backups, and processes for ensuring business continuity and disaster recovery. The cut-over process can be parallel or abrupt — if it's parallel, the old and new versions of the software must be run together before all users are migrated to the new version. However, this step usually keeps the software running for many years. During this process, all stakeholders — including users — interact with the software. The developer maintains the software under a maintenance contract, which may need approval from other departments. Therefore, there is significant manual interaction during this step.

Deployment Step	Typical Activities	Extenuating Issues	Timeframe	Automated or Manual
Verification	Understand the software requirements document; get hardware and software sizing estimates; complete organization IT procurement datasheet	Involvement of consultants, lawyers, and other professionals; involvement of corporate IT audit and risk management team	Few days to months (depending on the scope of software deployment)	Manual
Negotiation	Legal agreement with software developer; cost and delivery with hardware and support software vendor (the software to be deployed has already been negotiated by the customer); service-level agreements	Involvement of consultants, lawyers, corporate procurement, IT audit, risk management, and finance; multiple-country jurisdiction	Few days to months (depending on scope of agreements and cost)	Manual
Procurement	Actual shipment and delivery of all software and hardware	Shipment of specialized hardware or import from other countries	Few days to weeks	Manual
Installation	Of hardware and all software	Corporate checklists, external experts, corporate audit and review teams	Hours to days	Automated
Initialization	Hardware and software are initialized with data in the global settings including backup system settings; connect application with front-end (web and application servers) and backends (databases)	Multiple administrators for specialized domains (such as networking, database, NAS, security, etc.); integration with external systems such as emails and payment gateways	Hours to days	Manual
Configuration	Software is configured for use with user information and sufficient mass of data is populated in databases	Data migration, license activation, backup procedures implementation	Hours to days	Manual
User Acceptance Test (UAT)	Software is tested by users; automated testing tools may also be used for stress testing	Any defects found need to be fixed and depending on the severity the cycle of deployment process need to be revisited; user training	Days to months	Automated
Use	Actual production use of software; "going live"	Cutover - parallel or abrupt, helpdesk procedures, periodic user training, backups, business continuity, disaster recovery, maintenance contracts	years	Automated

Table 1. Typical Activities and Timeframes for the Generic Deployment Process

5. Conclusion

Software deployment is one of the most important phases in the software development life cycle, though the last one to occur before the software enters its maintenance phase. However, software deployment is anything but trivial and can last for months or even years depending on the complexity of the software. Improper deployment can lead to rejection of software by users, financial loss, or even litigation.

The software deployment process can be broken down into eight steps: verification, negotiation, procurement, installation, initialization, configuration, User Acceptance Testing (UAT), and production use. During verification, the software's deployment requirements regarding hardware and support software are understood. During negotiation, developers negotiate agreements and cost and delivery for deployment hardware and software. During procurement, the hardware and software needed for deploying the original software are procured. All software is installed

during the installation phase. During initialization, the software is prepared for subsequent steps. During configuration, user information is entered in the system and any integration with external systems are taken care of. During UAT, users test the software to ensure that it is ready for normal use. Any defects during this step are sent to the software developer for fixing. Finally, once the software passes the UAT, it is put to normal use.

Almost all of these steps require human involvement to a large extent. Only the installation, UAT, and actual use steps are mostly computerized. While automation can help reduce human effort in all eight of the steps, we are still a long way from completely automating the deployment process.

ABOUT THE AUTHOR

Nary Subramanian is an associate professor of computer science at the University of Texas at Tyler in Tyler, Texas. Dr. Subramanian received his Ph.D. in computer science from the University of Texas at Dallas. He is a Fellow of Service Learning at UT Tyler's Center for Excellence in Teaching and Learning. He has over 15 years' industry experience in engineering, sales and management. His research interests include software engineering, system engineering and security engineering.

Department of Computer Science
College of Business and Technology
University of Texas at Tyler
Tyler, Texas 75799
nsubramanian@uttyler.edu

REFERENCES

1. "2015 ERP Report." (2015.) Panorama Consulting Solutions. Available at http://go.panorama-consulting.com/rs/panoramaconsulting/images/2015%20ERP%20Report.pdf
2. http://searchsap.techtarget.com/definition/SAP-Rapid-Deployment-Solutions-SAP-RDS
3. Amin, P. (August 2012.) "What is the SAP Rapid Deployment Solution Implementation Methodology? Does It Work?" Available at http://scn.sap.com/community/rapid-deployment/blog/2012/08/06/what-is-the-sap-rapid-deployment-solution-implementation-methodology-does-it-work
4. Baumann, W. (April 2016.) "Courtroom lessons from a failed SAP ERP implementation." TechTarget. www.techtarget.com
5. Stein, J. (September 2014.) "FBI's Expensive Sentienl Computer System Still Isn't Working, Despite Report." Newsweek. http://www.newsweek.com/fbis-expensive-sentinel-computer-system-still-isnt-working-despite-report-272855
6. Wailgum, T. (September 2009.) "Why ERP Is Still So Hard." CIO. Available at http://www.cio.com/article/2424944/enterprise-software/why-erp-is-still-so-hard.html
7. Software Installation Process, IBM Knowledge Center. Available at http://www.ibm.com/support/knowledgecenter/ssw_i5_54/rzahc/rzahcswinstallprocess.htm
8. http://docs.openstack.org/developer/heat/template_guide/software_deployment.html#software-deployment-resources
9. http://www.tsg.com/resource-centre/articles/how-long-does-it-take-deploy-crm-system. Accessed Oct. 8, 2016.
10. https://docs.oracle.com/cd/E13222_01/wls/docs60/adminguide/config_web_app.html. Accessed Nov. 14, 2016.
11. http://www.oracle.com/us/products/applications/checklist-saas-2193759.pdf. Accessed Nov. 16, 2016.
12. http://searchmobilecomputing.techtarget.com/definition/mobile-device-management. Accessed Nov. 17, 2016.
13. https://docs.microsoft.com/en-us/intune/deploy-use/enroll-corporate-owned-devices-with-the-device-enrollment-manager-in-microsoft-intune. Accessed Nov. 17, 2016.
14. "General settings for Mobile Devices in Configuration Manager." (June 2015.) https://technet.microsoft.com/en-us/library/dn376523.aspx. Accessed Nov. 17, 2016.
15. https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html. Accessed Nov. 17, 2016.
16. http://www.iubenda.com/blog/the-need-for-privacy-policies-in-mobile-apps-an-overview/. Accessed Oct. 8, 2016.
17. Vemulapalli, A. & Subramanian, N. (November 2009.) "Transforming Functional Requirements from UML into BPEL to Efficiently Develop SOA-Based Systems." Lecture Notes in Computer Science. No. 5872, 337-349. Proceedings of On The Move 2009 Conference, edited by Meersman, R., Herrero, P. & Dillon, T.
18. Reeves, D. M., Wellman, M. P.,Grosof, B. N. & Chan, H. Y. (2000.) "Automated Negotiation from Declarative Contract Descriptions." American Association for Artificial Intelligence. http://www.mit.edu/~bgrosof/paps/kbem00.pdf
19. http://www.capterra.com/law-practice-management-software/. Accessed Oct. 9, 2016.
20. http://go.sap.com/product/srm/supplier-relationship-management.html. Accessed Oct. 8, 2016.
21. Krishnan, S. & Subramanian, N. (April 2015.) "Evaluating Carbon-Reducing Impact of Document Management Systems." IEEE Green Technologies (GreenTech) Conference. New Orleans, Louisiana.
22. Jacobs, D. "How to choose and implement automated configuration management tools." http://searchnetworking.techtarget.com/How-to-choose-and-implement-automated-configuration-management-tools. Accessed Oct. 8, 2016.
23. Vingrys, K. (July 2010.) "Acceptance Test Automation." https://www.thoughtworks.com/insights/blog/acceptance-test-automation. Accessed Oct. 8, 2016.
24. Carman, C. (November 2013.) "How to Manage User Acceptance Testing." http://insights.dice.com/2013/11/11/how-to-manage-user-acceptance-testing/. Accessed Oct. 9, 2016.

Why is Sprint Zero a Critical Activity?

Dick Carlson and Earle Soukup

Introduction. There is nothing new about project preparation. It doesn't matter how it is conducted, as long as the organization and the project team take some actions to ensure a majority of key obstacles are removed or mitigated. Have you seen projects halted – or worse, canceled – when progress was impaired to the point of utter chaos?

On Agile [1] or Scrum [2] projects, project preparation activities are often known as "iteration zero," "sprint zero," or "inception sprint." An investment in project preparation can vary depending on the complexity of the product, schedule constraints, the availability of skilled personnel and critical environments, and the amount of customer involvement. From our experiences, sprint zero activities for large teams are critical to ensure things go well from the start. A majority of our Agile work activities during the last 20-plus years has been with large, distributed teams that support government contracts in military weapon systems, avionics, mission planning, and satellite development. Small teams that are serious about paving the way to successful project execution perform some preparation, but not on the same scale as larger teams and projects. However, preparing for a project should not turn into a project itself, and while there is no set time for such preparation, budget and schedule constraints may be a major timing factor.

To be consistent herein with terms used in Scrum, we shall use the term "sprint zero" in this article. Also, the point of this article is to emphasize how vital sprint zero activities are to the success of any campaign, endeavor, mission, operation, program or project.

What is Sprint Zero?

The definition of sprint zero is vague. Its roots likely originated from costly lessons learned when things went awry during initial project execution. Typically used before an Agile or Scrum project commences, sprint zero has never, to our knowledge, been defined very well. The authors of this article believe the critical activities conducted by preparation or sprint zero are paramount to the success of any project. Such critical activities include the following:

- Identifying an Agile Champion.
- Writing some high-level user stories.
- Preparing an initial release plan.
- Training everyone on the benefits and use of Agile.
- Obtaining stakeholder buy-in prior to implementing any Agile method.
- Identifying a product owner (one per project).
- Identifying a Scrum Master (or Scrum Masters if multiple-teams will be employed).
- Identifying personnel that meet all requisite skill sets.
- Obtaining approvals from activity executives (who provide funding) and senior management (who provide personnel and other resources).
- Establishing all necessary infrastructures (e.g. test benches, racks, harnesses, initial architectures, Wi-Fi, and so on).
- Establishing all critical environments (e.g. test labs for operations, the software development area, team areas to support daily stand-ups, and so on).
- Creating just enough architecture to support initial sprints.
- Determining the most likely and the most significant risks.
- And more.

Many companies that teach Scrum do not mention sprint zero due to its unacceptable and controversial nature and its Scrum-like term. It could also be that sprint zero is not an Agile or Scrum activity, thus the resistance to the "sprint zero" term. The following are a few vague attempts to define sprint zero by both those who execute it and those who abhor it:

- Sprint zero is usually claimed as necessary because there are things that need to be done before a Scrum project can start.
- Sprint zero has three goals:
  1. Populate the product backlog with quality items.
  2. Provide a minimal environment that enables the writing of quality code.
  3. Write some code, no matter how small.

- Estimate the most important features, agree on a definition of "done," and rebuild confidence with the customer.
- The planning team is responsible for producing three deliverables by the end of the planning iteration:
  1. A list of all prioritized features or stories that include estimates.
  2. A release plan that assigns each feature or story to a sprint.
  3. A high-level application architecture, i.e., how the features will likely be implemented.

In April 2015, the author, Dick Carlson, posted a question to LinkedIn readers entitled, "Is Sprint Zero for You?" The purpose of the post was to determine how Sprint Zero is regarded by the Agile community and how many people use it. (You can read the post on LinkedIn at https://www.linkedin.com/pulse/sprint-0-you-dick-carlson?trk=mp-reader-card.) For those who are not LinkedIn members and do not have access to the post, portions of the post are included in the following section.

Is Sprint Zero for You?
The Original LinkedIn Post (April 14, 2015)

Watts Humphrey (1927–2010) once said, "If you don't know where you are, a map won't help." This may sound silly, but have you noticed how often excitement trumps logic when preparing for a new project? It would be interesting to know how many of you support sprint zero before project execution and how many of you avoid sprint zero altogether. Below are a few questions about the conduct of sprint zero, so please share your thoughts and opinions on this very important activity.

- *What is the potential impact that Agile adoption has on a company's operations, and which areas might be affected the most?*

- *Which critical foundational necessities must be completed during sprint zero to avoid major obstacles that might otherwise obstruct progress?*
- *Who should define all known requirements and create the initial product backlog prior to the start of a project?*
- *Who is responsible and the most capable person for creating a product vision, a product roadmap, and an initial release strategy?*
- *Who decides the amount of appropriate architecture that must be created to ensure that the completed functionality can be conducted successfully to ensure the team is building the right thing?*

(Latter portions of the LinkedIn post were removed from this article to avoid any perceived marketing pitches by the authors.)

The Reaction

Responses varied widely. Some supported sprint zero, while others had strong negative opinions regarding it. The majority of responses were received within a month. Since the reactions from readers were so mixed, a thorough analysis was needed. Earle Soukup, this article's co-author, took on the task.

The Analysis

The methods and results are explained herein, although an unspecified amount of readers may disagree with some of the conclusions.

Analytical Methods Used

The methods used for the analysis were partially quantitative and partially qualitative, but a portion of the quantitative analysis was subjective. The subjective aspect is derived from the difficulty of assigning a numerical value to someone's statement when its value is based on the opinion of the assessor.

Ground Rules

The ground rules for the analysis were simple:

- Each comment was assigned a value from -3 points (very negative) to +3 points (very positive). This includes a value of zero for some comments.
- A comment assigned a value of 0 was deemed to be inapplicable to the analysis. For example, a question or a request for an explanation may relate to the topic but not state an opinion. These comments were assigned a value of zero.
- A comment about any topic other than the subject of the LinkedIn post also received a zero value.
- A response of only a "Like" or a "Dislike" received a value of +1 or -1, respectively.
- A written response received a value of +/-2 or +/-3 depending upon its strength of support or rejection of the topics in the article.
- A longer written comment indicated a more intense opinion about the subject, so it generally received an assignment of +3 or -3.
- A comment containing an intense expression of opinion received a +3 or -3 value regardless of length. For example, "This is rubbish" received an assignment of -3.
- All comments or responses from the author, Dick Carlson, were excluded from the analysis and considered to be part of the original article.

The Results of the Analysis

The analysis revealed both strong (+/-2) and very strong (+/-3) opinions in the Agile- and Scrum-using community. Some respondents were of the opinion that there is an "orthodox" Agile, and any method that deviates from that orthodoxy is not Agile, nor can it be allowed to be classified as Agile. Other respondents were more flexible about using Agile.

Some interesting responses:

- Some respondents ignored the fact that Agile has expanded beyond the realm of developing software.
- Originally the application of Agile was to small, co-located teams, but now Agile is also being used by large, distributed teams.
- Some respondents seemed to reject the idea of evolution applying to Agile.
- Some respondents seemed to ignore one of the key values of the Agile Manifesto, "Individuals and interactions over processes and tools."
- Sometimes the agility associated with using Agile was ignored.
- Other respondents supported the ideas in the LinkedIn post, including the term "sprint zero."
- Yet other respondents supported the ideas in the LinkedIn post, but rejected the use of the term "sprint zero." Some respondents suggested terms would be appropriate to replace the term "sprint zero." Many suggestions had merit, but a few might introduce an ambiguity of the purpose for executing a sprint zero activity.
- One respondent expressed the opinion that the concept of sprint zero was apropos since there were zero deliverables at the end of the sprint—untrue, but interesting.

The Major Disagreement

The term "sprint zero" was the largest source of disagreement. The concept was anathema to some respondents. Their general opinion seemed to be that "a sprint must begin with a numeral, and it must produce some operating software at its end, or it is not a sprint." The implication of this statement is that only tested, accepted and executable code counts. No considerations were given to a project using Agile but developing something other than software.

Also rejected or ignored were the concepts that roadmaps, risk assessments, coding standards, and other work products that are valuable to project management. Company executives (who many people consider stakeholders), end-users, and other stakeholders do not count such as deliverable items. (Note: The aforementioned items are considered valuable to those holding the purse strings in a large corporation.)

Further Viewpoints

Other respondents liked the value of the activity but did not like the use of the term "sprint zero." Such comments were assigned values of +2, because a dislike for a particular term

seems less important than the value of the effort.

Many respondents agreed with the thesis of the LinkedIn post, either generally or wholly. In fact, on a weighted basis, more were in favor of the thesis in the LinkedIn post than opposed to it.

Some respondents drifted to other topics that were not germane to the subject of the LinkedIn post. These comments were assigned a value of zero (0), though some of these comments were rather interesting.

The Numerical Analysis

The numbers are revealing.

- Responses from the author were excluded, leaving a total of 171 responses from members of several LinkedIn groups, including the Agile & Lean Software Development Group, the Agile and Lean Europe Group, the Certified Scrum Master Group, the Non-Tribal Lean Agile Group, the Lean Agile Software Development Community, and LinkedIn Pulse contributors (who must be LinkedIn members to post such write-ups). The author's responses to comments made from respondents were explanatory in nature. (Note: The respondents included mostly software and system engineers from around the globe.)
- Neutral comments: questions, answers, or off the topic. 20 total responses.
- Comment: commenters provided only one or two comments each; one commenter provided at total of 23. Each comment was evaluated separately.
- Total Points: total, absolute value of the weighted comments was 286 points.
- The positive comments totaled 191 points; the negative comments totaled 95 points.
- The ratio of the positive to negative points is greater than 2:1.
- One discussion group provided 45 comments.

Conclusion

One might believe that for a technical project, the rules of logic would be the only rules that apply. But this analysis reveals that emotions may overrule the rules of logic; thus, emotions can influence technical decisions and opinions. Many people have strong emotions concerning Agile and how to apply it.

Some appear to believe that there is an "orthodox" approach to Agile, and any deviation from that approach prevents an activity from being called "Agile." Other users of Agile are prone to adjust their application to meet the needs of the situation. Their comments supported the approach that the emphasis should be on "meeting the needs of the situation" and not on establishing a process that followed "orthodox" procedures.

The responses from both sides of the issue demonstrated a strong support for Agile and Scrum, so the use of both methods should be viable for the future. Some individuals prefer to follow an "orthodox" approach while others are willing to apply Agile and Scrum in a flexible but comprehensive manner. Flexibility seems to outweigh rigidity by more than 2 to 1.

Emotional attitudes may have a stronger influence on the structure and behavior of a project than many people might believe. If the sprint zero term is offensive to you, then call it whatever you wish, but do not skip this step.

The distribution of responses is depicted in the following chart.

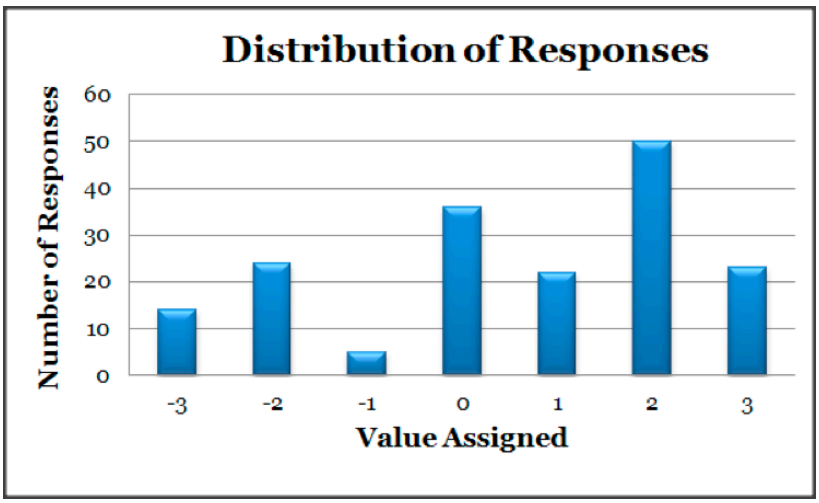


Figure 1. Distribution of Responses

As a final thought, essentially all training providers make it very clear that the team's Scrum Master has the authority to cancel a project should impediments and other daunting obstacles become burdensome and overwhelming.

NOTES

1. Agile Alliance. https://www.agilealliance.org
2. Scrum, Scrum Alliance. https://www.scrumalliance.org

ABOUT THE AUTHORS

Dick Carlson has a B.S. degree in business management and is certified as a Scrum Professional, Scrum Master, Scrum Product Owner, and in Lean-Agile Project Management. He has shared successful experiences of Agile, Lean, and Scrum implementations at conferences, workshops, and symposia. Dick's engineering career spans 50 years, and he has taught courses in mathematics, electronics, CMMI, configuration and data management, Agile, Lean, and Scrum for more than 30 years.

Mr. Soukup holds a Bachelor's and Master's degree in Electrical Engineering, a Juris Doctor in Law, certificates for software development, project management, functional management, systems engineering, and Agile and Lean including being a Certified Scrum Master. He was a development and test engineer, manager, and project manager for both hardware and software. Also he developed and taught courses in mathematics, electronics, ethics, Lean, and Agile. He is an accomplished analyst.

Upcoming Events

Visit <http://www.crosstalkonline.org/events> for an up-to-date list of events.

Software Solutions Symposium 2017
Arlington, Virginia
20-23 March 2017
http://www.sei.cmu.edu/sss/2017/

15th Annual Conference on Systems Engineering Research
23-25 March 2017
Redondo Beach, CA
http://viterbi.usc.edu/sae/cser2017.htm

Design, Automation and Test in Europe
Lausanne, Switzerland
27-31 March 2017
https://www.date-conference.com/

2017 IEEE Third International Conference on Big Data Computing Service and Applications (Big Data Service)
San Francisco, CA
6-9 April 2017
http://big-dataservice.net

Cyber-Physical Systems
Pittsburgh, PA
18-21 April 2017
https://cpsweek2017.ece.cmu.edu/

Mobile Dev + Test Conference
24-28 April 2017
San Diego, CA
https://mobiledevtest.techwell.com/

Software Engineering Institute (SEI) Architecture Technology User Network Conference (SATURN) 2017
Denver, Colorado
1-4 May 2017
http://www.sei.cmu.edu/saturn/2017/

ACM CHI Conference on Human Factors in Computing Systems
Denver, Colorado
6-11 May 2017
http://chi2017.acm.org/

39th International Conference on Software Engineering
20-28 May 2017
Buenos Aires, Argentina
http://icse2017.gatech.edu/

IWPE 2017 : 3rd IEEE International Workshop on Privacy Engineering
25 May 2017
San Jose, CA
http://ieee-security.org/TC/SPW2017/IWPE

ENCASE 2017: 12th International Conference on Evaluation of Novel Approaches to Software Engineering
28-29 April 2017
Porto, Portugal
http://www.enase.org/

12th Annual System of Systems Engineering Conference
18-21 June 2017
Waikoloa, Hawaii
http://sosengineering.org/2017/

10th IEEE International Conference on Cloud Computing
25-30 June 2017
Honolulu, Hawaii
http://www.thecloudcomputing.org/2017/program.html

28th Annual IEEE Software Technology Conference
25-28 September 2017
Gaithersburg, MD
www.ieee-stc.org

Whole Lotta Shakin' Going On!

(with apologies to Jerry Lee Lewis)

Come on over baby, whole lotta processes goin' on
Yes I said come on over baby, baby you can't go wrong
We ain't fakin' a whole lotta processes goin' on

I have to admit – I am not a young man anymore. In fact, unless I plan on living into my 120s, I'm not even "middle aged". As a college professor, I promise each class that they will not have to hear more than five "Back when I was your age…." stories per class. (OK – truthfully, I tell them ten). It's hard not to tell those stories – I don't think you can fully appreciate modern technology and processes unless you understand the way "it used to be done". And our profession is so young. Engineers have been building bridges, great walls, and pyramids for thousands of years. Doctors has been practicing medicine a long time – the Hippocratic oath dates from the fifth century BCE. Lawyers and politicians have been around …… never mind.

How long have software developers been around? Well, seventy-five years ago – we were using slide rulers, and we were happy to see electronic calculators (invented in 1972?) Doctors talk about blood-letting in their history, and engineers talk of building bridges with just rock and mortar. Software developers? Younger ones talk about "OMG – I actually had to code a program in FORTRAN!!"

Back in the 1970s, I was an instructor in the basic developer course at Keesler AFB, Mississippi. We taught a basic programming course using, as I recall, a Hughes 407L (although I can find no reference to this computer – my memory might be slipping. Feel free to email if you have a better memory.) Granted – it was an old computer – but still adequate for teaching basic concepts. We used it to teach assembly language programming. To the best of my memory, listed below are the steps involved in running a program. Note that this was only 45 years ago!

0. Before you started, of course, you had to punch your program onto a card deck. We used an IBM 029 model.
1. Take your card deck with you into the computer room during your reserved 30-minute slot. Kick out the developers using the computer, ignoring their "Just 5 more minutes?" please.
2. Power down the computer (which was room sized!) and power it up, to ensure clean memory.
3. Locate the "Operating System" card deck from the card shelf, and place it in the card reader.
4. Go to main panel, and press the "Boot Init" button, which would load the OS deck, and execute the code. OS now running. Return OS deck to card shelf. Locate the Assembler deck, and place it in the card reader.
5. Go to the main console, and type "load/run". This caused the OS to read the Assembler card deck into memory, and execute it. The assembler was now ready for input.
6. Place the Assembler deck back. Load your program into the card reader, go to console, and type "continue". You card deck (your program) was now loaded as data into the assembler, and the assembler, well, "assembled" it.
   a. If there was an assembly error – you got a listing of the error at the printer. You then frantically tried to fix it during your 30-minute slot, and restarted from step 2 above.
   b. However, IF your program had no errors, the card punch produced an object deck – ready to be linked and loaded.
7. Grab the "Link and Load" card deck, add your object card deck to the end, place it in the card reader, and type "load/run" on the console. The "Link and Load" program linked in code for system routines, created an executable module in memory, and executed it.
   a. IF the program ran successfully, your output showed up on the console. Tear off the paper from the console (it was a teletype, NOT a CRT monitor) and you were done!
   b. On the other hand, if an ABEND (Abnormal ENDing) occurred (i.e. the program crashed) you went to the console, typed "dumpmem/all" and retrieved a dump of ALL 32K (!) of memory on the line printer. Try and pour through memory to find error. Go to step 6b above.

Processes are MUCH simpler now. But – you know what? We have forgotten the art of desk-checking code in the last 40 years. The pain and difficulty of the steps above guaranteed that you didn't just casually type up a deck of cards without seriously reading (and re-reading) looking for typos. However, the great part of desk-checking was that you found both syntax AND semantic (logic) errors as you read through your code.

Nowadays, compiles are so quick and easy (typically, "push one button") that we no longer desk-check for syntax ("let the compiler do that!") And, sadly, we seldom desk-check for semantics until we try and run the code. We have forgotten how to individually review code for semantics. We have IDEs (Integrated Development Environments) that compile as we are coding – and fix syntax errors as we type. Why bother to desk-check?

Back in the 1970s, we had a simple process for writing code – it was called the "code and fix" process. Basically, it was a "repeat until somebody gives up" process.

Nowadays – NOBODY uses the "code and fix" process model, right? (Pause for sarcastic and guilty laughter). We have developed more complex and better processes that produce software that is more reliable, understandable, modifiable/maintainable and efficient. The processes improve quality, increase productivity, and reduce wasted time fixing the same error over and over …

But remember that a process does not replace creativity, imagination, and thinking. There is even more of a need to desk-check code. Back in the 1970s, a 2000 line program was considered huge. Now? 7 million lines of code is relatively common. Rather than just reading 2000 lines of a single program, we now need to review the code and the design of a tightly coupled 7 million line, 100+ program system. In fact, we have to review multiple components of the design: architectural, data, interface, and finally module (the code).

There has never been a greater need for good processes. Likewise, there has never been a greater need for developers who understand the process, and use their skills and intelligence and experience to keep alive the spirit of desk-checking. Don't let the process take the place of individual reviews and common sense.

It could be worse. You could still be looking for a FORTRAN compiler card deck to load into a card reader. And "keep on shakin' ".

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