Assessing DoD System Acquisition Supply Chain Risk Management
Defense capabilities are supported by complex supply chains. This is true for weapons systems and large “systems of systems” that enable force projection and also true for service supply chains. Important requirements for both capabilities now depend on the cybersecurity and related assurance level of third parties that also come with their own risks.
By Christopher Alberts, John Haller, Charles Wallen and Carol Woody

Issues in Development and Maintenance of Open Architecture Software Systems
This article identifies and describes six emerging issues that affect the engineering of open architecture software systems that integrate proprietary and open source software components.
By Walt Scacchi, and Thomas Alspaugh

The Founding Fathers of Agile
Pioneers and innovators in their own fields, John D. Rockefeller and Alexander Hamilton’s legacies are also early examples of Agile project management.
By Peter Morris

Software Process Improvement Standards and Guides for Very Small Organizations: An Overview of Eight Implementations
It has been established that Very small entities (VSEs) — organizations with up to 25 people — often do not utilize existing best practice standards and frameworks such as ISO/IEC/IEEE 12207 software life cycle processes standard. In this paper, we present eight implementations of ISO/IEC 29110 as an exemplar of the potential benefits from the use of this standard.
By Claude Lapore and Rory O’Connor

Leveraging the Power of Historical Data through the Use of Trend Lines
A recent initiative put forth by the DoD attempts to improve its cost estimation process by leveraging historical data collected from forensic analyses of recently completed software development efforts. This article discusses some of the challenges faced throughout this initiative, the data collection process, and how one can leverage data to improve cost estimates.
by Taylor Putnam-Majarian and John Staiger

Debunking Agile Myths
This article discusses and analyzes 10 of the most common misconceptions about Agile practices and techniques and identifies real-world examples that debunk these misconceptions.
by Dick Carlson

Open Forum: Which Agile Method is Best?
People have a tendency to want the “best” of something. It is a valid question, and it has a clear and emphatic answer: “It all depends.”
By Earle Soukup
FROM THE SPONSOR

CrossTalk would like to thank NAVAIR for sponsoring this issue.

"Most of the effort in the software business goes into the maintenance of code that already exists."

—Wietse Venema, Dutch programmer, physicist, and author of the Postfix email system

Operations and Maintenance. Welcome to the May/June 2017 edition of CrossTalk. The development and procurement of new weapons, platforms, and capabilities is not the most expensive part of acquisition within the Department of Defense: it is the operations and maintenance of those systems and capabilities over their lifetimes. For the physical systems, those costs are fairly well known and predictable. For the software upon which the DoD depends, that picture is not as clear.

With the recent history of tight military budgets and an increasing pace of military operations, the DoD has had to re-allocate funding to maintain the operational pace of the War on Terror. Necessary expenditures and activities have been prioritized, with those of a lower priority being deferred in order to support the near-term mission. The effect of that in the long-term for the DoD can be extrapolated from statements made recently by Vice Chief of Naval Operations Admiral William Moran on January 11th 2017 at the Surface Navy Symposium in Arlington, Virginia. At the event he said “Deferred maintenance is insidious, it takes a toll on the long-term readiness of the fleet…” “When the transition team came around to all of us in the building and asked us what we could do with more money right now, the answer was not to buy more ships. The answer was to make sure the 274 that we had were maintained and modernized to make 275 ships worth of combat power…”1 This issue might be considered an inconvenience or even a hazardous condition in a fight where our forces hold an overwhelming advantage over the enemy. In the event of a conflict between peer-level nation states it could mean the difference between victory and defeat.

A similar influence has been seen in the operations and maintenance of DoD software. The high demand for new software capabilities to meet the challenges of the changing battle space environment has resulted in an informal reprioritization of software development activities. Efforts to remove known defects from fielded software systems are often given a low priority in order to get new capability to DoD forces quickly. The new capabilities being developed and fielded have been impacted as well; as software developers, in the name of speed, are often allowed to cut corners in software quality processes, documentation, and other activities that would make those systems more usable, reliable, and maintainable. In the near-term, the operators of DoD software systems rely on workarounds to mitigate the impact of non-critical defects, and DoD software maintainers are faced with code that is not well documented and which contains an unknown number of preventable defects. This might be an inconvenience today, but the effect on operations and maintenance in a peer-to-peer conflict could be serious.

In order to prepare for emerging threats, the DoD is facing an expensive and lengthy effort to correct the shortfalls in critical capabilities caused by years of shifting operations and maintenance priorities. The world of DoD software systems is facing a similar challenge. Fortunately, this edition of CrossTalk provides the reader with some tools to assist in identifying areas of risk, improving the ability to predict the cost and schedule impact of software development and maintenance, and in adopting processes that will deliver better engineered software.

David Saint-Amand
Lifecycle Logistician with the NAVAIR Process Resource Team
Assessing DoD System Acquisition Supply Chain Risk Management

By Christopher Alberts, John Haller, Charles Wallen, and Carol Woody

Abstract. Defense capabilities are supported by complex supply chains. This is true for weapons systems and large “systems of systems” that enable force projection — for example, a weapons system like the F-35 Fighter. It is also true for service supply chains — for example, the array of private logistics firms that the Department of Defense (DoD) relies upon to transport personnel and equipment around the world. Important requirements for both capabilities (force projection and transportation) now depend on the cybersecurity and related assurance level around the world. Important requirements for both capabilities (force projection and transportation) now depend on the cybersecurity and related assurance level of third parties. While supplier, vendor, and contracts relationships provide cost savings and flexibility to the DoD, they also come with risks.

Supply chain cyber risks stem from a variety of dependencies, in particular from the processing, transmittal, and storage of data, and on information and communications technology. These cyber risks in the supply chain are broad and significant. Important mission capabilities can be undermined by an adversary’s cyberattack on third parties, even in situations where the government is not explicitly contracting for technology or services like data hosting. In one sense, this risk stems from the reality that virtually all products or services that the DoD acquires are supported by or integrated with information technology.

Some recent DoD efforts have focused on mandating contractual standards for specific technical controls. For example, the confidentiality of controlled unclassified defense information shared with contractors has long been a concern. To address this problem, DoD recently modified the Defense Federal Acquisition Regulation Supplement (DFARS) subpart 252.204. Section 252.204-7008 mandates that contractors must institute the controls listed in NIST Special Publication 800-171 to protect “covered defense information” [DFARS 2016]. After much discussion and some consternation in the defense contracting community, DoD subsequently delayed implementation of the revised rule until December 2017. Regardless of the specific timetable, focusing on technical controls in place at the time of contracting is only part of the solution.

A top-down, one-size-fits-all approach to formal acquisition requirements may not account for situations where the appropriate requirements differ from the mandated, specific controls. For example, the controls for private companies that were specified in NIST Special Publication 800-171 were selected for their relevance to the confidentiality of information [Ross 2015]. These controls may not be adequate in situations where the defense capability relies on the availability of systems. In military transportation, for example, this may include the availability of an air traffic control system or a freight forwarding and tracking system.

Defense organizations may be constrained in their choice of contractors. One contractor may be the sole source for a specific, vital capability. Finally, while contractual terms can incentivize and provide guidance to contractors, they hardly form a basis for assurance. In many cases, these controls may only be effective at a single point in time. The government might not be able to tell whether or not a control was actually being maintained.

Cyber risks originating in the supply chain can be made worse by slow internal processes. For example, software-specific issues arise when vendors issue critical security patches for products that reside on classified networks. The lengthy time needed to recertify the software in delays in applying these patches, leaving systems unnecessarily vulnerable.

Defense organizations need a range of capabilities to manage supply chain cyber risks. These capabilities typically involve additional staff beyond those who are assigned to acquisitions. Leaders should prepare program offices and sustainment depots to address supply chain cyber risks by enhancing their ability to manage supplier performance across a range of capabilities, important practices, and processes.
This article presents a framework of supply chain risk management practices to measure and improve an organization's ability to manage third-party cyber risks. This framework is not a new regulation or policy. Instead, it provides a mechanism for increased confidence about the current level of vendor performance, a fuller understanding of gaps, and a road map for improvement.

Cyber Risks in the Defense Supply Chain: A Growing Challenge

The growing DoD reliance on complex, sometimes global supply chains to deliver military, civil and intelligence capabilities has increased cybersecurity concerns. This complexity makes it more challenging than ever for acquirers to understand, monitor and manage supply chain products and processes.

Managing supply chain cyber risk is especially challenging because that risk is broad and pervasive. The National Institute for Standards and Technology (NIST) defines the Information and Communications Technology (ICT) supply chain as a "linked set of resources and processes between acquirers, integrators, and suppliers that begins with the design of ICT products and services and extends through development, sourcing, manufacturing, handling, and delivery of ICT products and services to the acquirer" [Boyens 2012]. However, the related risks often extend to areas which are not explicitly related to information technology, including the following:

- Manufacturing and integration supply chains: Responsible for conceptualizing, designing, building and delivering systems and hardware.
- Service supply chain: Responsible for providing services to acquirers. In a defense context, this includes services that vary as widely as data processing and hosting, logistical services, and support for administrative functions.
- Software supply chain: Responsible for producing the software that runs on vital systems.

Each of these supply chains differs in their output, or what they provide to a defense acquirer. As a result, certain vulnerabilities are frequently associated with specific types of supply chains. The software supply chain may introduce vulnerabilities involving open source software modules, such as the Heartbleed security flaw [1]. In addition, there may be risks involving unintended functionality of a particular piece of software. However, DoD faces a similar problem when it comes to each type of supply chain. The security of critical capabilities depends on how well organizations outside direct DoD control address the confidentiality, integrity, and availability of assets that DoD relies upon. Indeed, no particular type of risk is confined to one type of supply chain. DoD must have confidence that effective security and resilience have been built into key systems and services, as well as confidence that they can be sustained. Failures in any of the supply chains that DoD relies on can lead to:

- A breach of confidential information shared with a supplier or vendor.
- A lack of availability of critical service or capability.
- Problems with the integrity of data or systems.
- An attacker maliciously tainting or corrupting the output of the supply chain.

Supply chains are characterized by the need to manage external entities through arm's-length contracts. Unfortunately, the government has a limited ability to verify that good engineering and cybersecurity practices are actually in place. The DoD’s ability to properly manage and mitigate these risks should not exclusively rely on either initial contracting processes or the ability to validate controls at the supplier. A framework to help measure, manage and improve DoD capabilities would help to ensure that supply chain risks can be mitigated.

Software: A Case Study in Supply Chain Complexity

To illustrate some of the issues and complexities inherent in managing supply chain cyber risks, we examine the software supply chain as a case study. Software is increasingly taking over functionality previously handled by hardware components. The percentage of system functions performed by software has risen exponentially in recent years. Key benefits include increased flexibility in development schedules and rapid response to changing needs.

However, we are now faced with a situation where the capabilities of today's software technology environment, the need to outsource, and the interaction between off-the-shelf and open source software products have far outpaced our ability to effectively monitor and manage the risk using traditional methods. With the critical roles software holds in our operational environments, the impact of fakes, frauds, and malicious activities could be devastating.

Today’s software is riddled with weaknesses and vulnerabilities that represent unacceptable security risks. [McGraw 2006, 3]. The increasingly global nature of software development has also raised concerns that global supply chains could be compromised, allowing malicious code to be inserted into a delivered software product during development or enabling a compromised product to be substituted during delivery or installation.

Like any supply chain, the ownership and responsibility for software may be scattered across several organizations, each of which may affect the security and integrity of the final product. For example, software is often assembled from existing code (see Figure 1 for a hypothetical example). An application may be made up of components from commercial products, open source, and “glue code” that ties these various pre-existing pieces together to provide the desired functionality. Each of these components can also be made up of multiple pieces, and the supply chains can be long with many diverse sources.

Figure 1
Software Assembled from Existing Code Components

As software is assembled and integrated into a final product, each integrator inherits all of the vulnerabilities inserted by the developers at every sub-level, which can include additional custom, commercial, open source, and legacy software. Software flexibility that allows system users to adjust functionality to meet changing operational needs also provides this same capability to an attacker. In addition, suppliers are often free to update or make changes to the software as needed, which may impact customers (licensees) in unforeseen ways.

The software supply chain comes with an additional complication. Depending on the specific business agreement, an acquirer may have a very limited ability to verify the quality or integrity of the supplied software. A commercial software provider allows use of their product through a licensing arrangement but does not give away the source code. Open source software is frequently also controlled through licensing but with a very different relationship structure: access to the source code is provided and copies of the code can become different products using the original baseline. An organization may also acquire a service which is based on software but the service provider, not the organization, owns the software[2].

The nature of software products has also evolved in recent years with the emergence of computer networks. The focus has shifted from producing stand-alone software products to providing technical capabilities within a larger “system-of-systems” context. A “system-of-systems” is defined as a set or arrangement of interdependent systems that are related or connected (i.e., networked) to provide a given capability. An organization may also acquire a service which is based on software but the service provider, not the organization, owns the software[2].

A common way to control risks in software acquisitions is a formal contract with a supplier. The contract governs the relationship between two organizations and includes a set of requirements. The supplier develops a software product that meets those requirements. Another example of a formal agreement between organizations is when an acquirer licenses commercial off-the-shelf (COTS) software from a supplier. A license outlines any terms and conditions regarding the acquirer’s use of the software product.

However, a variety of factors have increasingly made the formal legal contract less effective in terms of mitigating risk in software acquisitions. The complexity of supply chains means that suppliers may have a limited ability to verify that each component of a final product meets the requirement. Software is frequently built for many uses and it can be difficult to determine if it is really “fit for use” in a specific case. Suppliers may be subject to very sophisticated nation-state attacks. Finally, it may be very difficult for the government to prove that a software defect or problem was really the problem in specific cases. This goes directly to a basic question about any contract: How will it be enforced?

Specific technical requirements and contracts; these are the two primary means with which the government has attempted to control cyber risks in the supply chain in the past. While these are basic and foundational methods to control supply chain risk, they increasingly do not suffice. For the government to realize a sufficient level of assurance will require attention to the processes and capabilities employed internally and by key contractors.

Toward Enterprise Management of Supply Chain Cyber Risks: Acquisition Security Framework (ASF)

Commercial, state and federal acquirers have shifted much of the responsibility for addressing supply chain cyber risks in acquisition to integration contractors, software product vendors, and service providers. In many cases, the government will also rely on a prime contractor to sustain systems after the acquisition is complete, taking on the role of directly maintaining the software components and handling this through a service supply chain. This reliance on an intermediary — a prime contractor or integrator — means that the final integrity and suitability of the system depends in large part on how well and maturely the government and prime contractor manage supply chain cyber risks.

The Acquisition Security Framework (ASF) is a way to assess and improve these capabilities. The use cases for the ASF include assessing the capability of relevant organizations to identify and remediate gaps in supply chain risk management capability. While it can be employed as part of due diligence to assess a prime contractor before contract award, its usefulness is not limited to that application. A framework like the ASF could enable groups of contractors to identify and make improvements consistently and collectively for the benefit of an entire acquisition project or community of suppliers. This type of use requires the free and protected exchange of information that concerns not only threats and incidents, but also potentially sensitive information about the practices and internal governance that a contractor has in place.

The ASF was developed from previous work at Carnegie Mellon University’s Software Engineering Institute (SEI). In 2014, the SEI’s CERT Division developed the External Dependencies Management (EDM) Assessment. It focuses on the capability of critical infrastructure organizations in the United States to manage external dependency and supply chain risks and is an extension of the DHS Cyber Resilience Review (CRR)[3] The EDM Assessment is based on the structure of the CRR and the CERT Resilience Management Model.[4] It provides private sector organizations with a lightweight way to measure their ability to manage this type of risk.

Recently, CERT started investigating how to apply concepts from the CRR and EDM Assessment to help defense organizations evaluate risks and gaps in how they manage supply chains. The initial product is this draft ASF, which is intended to assess supply chain processes for acquiring, engineering and deploying secure software-reliant systems. One application area of particular interest is assessing supply chain processes for DoD weapon system acquisition programs. Focus on such programs will be a priority for this work moving forward.

Risk Factors for Software Supply Chains

Development of the ASF also incorporates an additional, foundational piece of SEI supply chain research. In 2010, an SEI team conducted an applied research project focused on evaluating and mitigating software supply chain security risks across the acquisition life cycle. [Ellison 2010] The 2010 research project led to the identification of the following risk factors for software supply chains in particular:
Supplier capability: Ensuring that a supplier has good security development and management practices in place throughout the life cycle.

Product security: Assessing a completed product’s vulnerability to security compromises and determining critical risk mitigation requirements.

Product distribution: Ensuring that the methods for delivering the product to its user are secure and determining how these methods guard against the introduction of malware while in transit.

Operational product control: Ensuring that configuration and monitoring controls remain active as the product and its use evolve over time.

### Framework of the ASF

We designed the ASF to build upon the content and structure of the EDM Assessment to the extent possible. We also designed it to address the four software supply chain risk factors identified in the previous section. The resulting framework comprises five practice areas:

1. Relationship Formation.
2. Relationship Management and Governance.
3. Engineering.
5. Supply Chain Technology Infrastructure.

In the rest of this section, we describe the critical topics to be addressed in each practice area and map them to the supply chain risk areas.

### Relationship Formation (Practice Area 1)

Relationship Formation focuses on acquirer practices for evaluating and controlling supplier-related risks before the acquirer enters a relationship with its suppliers. This area also includes practices for contracting with suppliers. Much of the content for this area comes from the corresponding area of the EDM assessment. The following topics are covered in this area of the framework:

- Planning relationship formation.
- Risk management process.
- Supplier capability evaluation.
- Cybersecurity in formal agreements (and in the contracting process).

### Relationship Management and Governance (Practice Area 2)

Relationship Management and Governance defines practices for managing ongoing relationships with suppliers. We make heavy use of content from the corresponding area of the EDM assessment when developing practices for relationship management and governance. The following topics are covered in this area of the framework:

- Supplier identification and prioritization.
- Supplier risk management.
- Supplier performance governance and management.
- Change management.
- Supplier transitions.
- Management of supplier access to acquirer assets and technologies.

### Engineering (Practice Area 3)

Engineering comprises practices to build appropriate cybersecurity controls into a weapon system and minimize the chance of accidentally inserting vulnerabilities. The following topics are covered in this area of the framework:

- Requirements.
- Architecture.
- Development/coding.
- Integration.
- Testing.
- Independent validation and verification (ensuring product security).
- Deployment (including product distribution).

### Secure Product Operation and Sustainment (Practice Area 4)

Secure Product Operation and Sustainment includes practices for managing cybersecurity risk as software-reliant systems are operated and maintained over time. The following topics are covered in this area of the framework:

- Documentation that supports secure product configuration.
- Tools that supports secure product configuration.
- Product updates for security (e.g., software patches).
- Engineering and technical support for managing cybersecurity incidents.

### Supply Chain Technology Infrastructure (Practice Area 5)

Supply Chain Technology Infrastructure includes practices for securing technologies that support the execution of supply chain activities. The following topics are covered in this area of the framework:

- Management of acquirer’s systems and networks.
- Disruption planning.
- Maintaining and updating control requirements for suppliers.
- Situational awareness requirements for suppliers.
- Procurement of technology (e.g., COTS, open source software, hardware).
- Management of infrastructure dependencies.

<table>
<thead>
<tr>
<th>Supplier Capability</th>
<th>Product Security</th>
<th>Product Distribution</th>
<th>Operational Product Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Relationship Formation</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Relationship Management and Governance</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3. Engineering</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4. Secure Product Operation and Sustainment</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5. Supply Chain Technology Infrastructure</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

**Table 1. Mapping of Practice Areas to Risk Factors**

### Mapping ASF Practice Areas to Risk Factors

Table 1 shows how the five practice areas of the ASF framework map to the four software supply chain risk factors defined earlier in this article. Relationship Formation (Practice Area 1) and Relationship Management and Governance (Practice Area 2) both address supplier capability. Product security and product
distribution are included in Engineering (Practice Area 3). Secure Product Operation and Sustainment (Practice Area 4) provides insight into operational product control. Finally, Supply Chain Technology Infrastructure (Practice Area 5) includes cross-cutting practices that apply to all four risk factors.

The ASF is currently a prototype based on our initial study of software supply chains. It does not represent a conclusive solution to the problem of cyber risks in supply chains. Considerable development and iteration are needed to produce a final baseline version of the ASF framework that includes all relevant cybersecurity practices.

Conclusion

Cyber supply chain risk management has not been broadly and effectively applied across DoD system acquisition and development life cycles. The DoD has already experienced supply chain-related cyber incidents that affected critical systems and capabilities. It also does not appear to have a repeatable, consistent way to measure and mitigate these types of risks across the enterprise. The DoD’s efforts to manage this problem have generally involved the inclusion of requirements and specific technical controls in the complex contracts used to govern supply chain relationships. Functional, technical, and security requirements—as well as formal contracts—are certainly elements of supply chain risk management. However, by themselves they cannot provide the speed, flexibility, and repeatability required to manage risks across the entire DoD enterprise and contractor community.

We believe the best predictors of resilience and security across an acquisition are the completeness, efficiency, and institutionalization of the right practices in the relevant organizations. A range of effective practices can be drawn from the management of operational risk in privately owned critical infrastructure. This article identifies a starting point using five operational practice areas that map to high-level risk factors identified in earlier analyses of cyber risks in software supply chains. Further development and use is required to refine this approach. However, through partnership and collaboration, it should be possible to increase the sustainability, effectiveness, and agility with which DoD and industry partners identify, mitigate, and respond to cyber security threats in critical supply chains.

Acknowledgement

Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Homeland Security or the United States Department of Defense.

NO WARRANTY, THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN *‘AS-IS* BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-U.S. Government use and distribution.

Carnegie Mellon® and CERT® are registered marks of Carnegie Mellon University. DM-0004048

NOTES

2. For example, at this time, we do not have proven techniques that can find all malicious code that has been inserted into a software component. Part of the difficulty is that in addition to viruses, worms, and Trojan Horses, malicious code can consist of subtle changes that create a vulnerability that an attacker could exploit when the software is deployed. The intent of maliciously inserted code is to change software behavior.
3. https://www.us-cert.gov/ncParticipants/assessments
Christopher Alberts is a Principal Engineer in the CERT® Division at the Software Engineering Institute, where he leads applied research projects in software assurance and cybersecurity. His research interests include risk analysis, measurement, and assessment. He has published two books and over 40 technical reports and articles. Alberts has BS and ME degrees in engineering from Carnegie Mellon University. Prior to the SEI, he worked at Carnegie Mellon Research Institute and AT&T Bell Laboratories.

John Haller is a member of the technical staff on the Cybersecurity Assurance team within the CERT® Division at the Software Engineering Institute (SEI), a unit of Carnegie Mellon University in Pittsburgh, PA. As a member of this team, Haller performs research on critical infrastructure protection, focusing on methods, tools and techniques for managing external dependency and third party risk. Prior to joining CERT in 2010, Haller was analyzing cybercrime attacks on the financial industry in collaboration with a U.S. law enforcement agency. Haller, a U.S. Army veteran, received his Juris Doctor (cum laude) and Master in Public and International Affairs from the University of Pittsburgh and is also a Certified Information Systems Security Professional (CISSP).

Charles M. Wallen has been a thought leader in operations and IT risk management for over 20 years. He has provided consulting to public and private organizations, led industry-wide initiatives, and managed global operations risk management and governance programs at American Express and Bank of America. Charles works closely with the CERT Division of Carnegie Mellon University’s Software Engineering Institute as a Senior Member of the Technical Staff. His work at CERT focuses on resilience management, external dependency risk management, and critical infrastructure protection. He is a Principal with Spectrum Consulting Services, in Dallas, Texas, which he formed in 2004. Spectrum provides management consulting to a variety of industries, specializing in operations risk management, cyber security, business continuity, supplier oversight, and governance.

Dr. Carol Woody is the technical manager of the CERT Cyber Security Engineering team at the Software Engineering Institute (SEI) at Carnegie Mellon University. Her research focuses on software engineering lifecycle capabilities to measure, manage, and acquire software and systems with operational cyber security. She has co-authored a book, Cyber Security Engineering: A Practical Approach for Systems and Software Assurance, which assembles proven techniques for building software systems effectively.

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:

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>. 
We identify six kinds of emerging software life cycle challenges that we have observed within the U.S. Defense community as they have moved to OA systems for C2 that integrate, operate and maintain contemporary OSS and CSS components. These challenges follow from our ongoing efforts within the Defense community that build on and refine our previous efforts in this area, [14, 15] and point to areas of practice requiring further software engineering (SE) research.

Unknown or Unclear OA System Representations

This first kind of challenge arises when acquiring new or retrofitting legacy software systems that lack an open or explicit architectural representation. Such a representation should identify and model major software system components, interfaces, interconnections and remote services (if any). Though OA reference models are in use within the SE research community, contemporary C2 generally lack such descriptions or representations that are open, sharable, or reusable. This may be the result of legacy contractor business practices that see software architectures as proprietary intellectual property (IP), even when OSS components are included or when applications’ subsystems are entirely made of OSS code. An alternative explanation reveals that complex software systems like common web browsers (Mozilla Firefox, Google Chrome, Apple Safari, Microsoft Edge) have complex architectures that integrate millions of SLOC that are not well understood and that entail dozens of independently developed software elements with complex APIs and IP licenses that shift across versions. [12] This implies the effort to produce an explicit OA reference model is itself a daunting task of architectural discovery, restructuring, and continual software evolution. [5, 7] Thus, new ways and means for extracting software components, interconnections, and interfaces and transforming them into higher-level architectural representations are needed.

Heterogeneously licensed OA systems

OSS components are subject to widely varying copyright licenses, end-user license agreements, digital civil rights, and other IP protections. The Open Source Institute recognizes dozens of OSS licenses that are in use, though the top 10 represent more than 90 percent of the open source ecosystem. [9] This is especially true for OSS components or application systems that incorporate source code from multiple independent OSS development projects, such as those found in contemporary web browsers like Firefox and Chrome that incorporate components from dozens of OSS projects, most with diverse licenses. [12] This means that OSS application systems are subject to complex software IP obligations and rights that may defy tracking or entail contradictory obligations/rights. [1] Determining overall IP obligations for such systems is generally beyond the scope of expertise for software developers, as well as most corporate lawyers. Furthermore, we have observed many ways in which IP licenses interact within an OA software system, such that different architectural design choices that configure a fixed set of software components result in different overall system obligations and rights. Understanding multiple license interactions and IP mismatches is far too confusing for most people and a source of legal expense or, alternatively, expensive indemnifica-
tion insurance policies, for the software producers or system integrators. Nonetheless, in our view, OA software ecosystems are defined, delimited, and populated with niches that locate specific integrated system solutions. [12] Furthermore, we see that these niches effectively have virtual IP licenses that must be calculated via the obligations and rights that propagated across integrated system component licenses via union, intersection, and subsumption relations among them. [4] Such calculation is daunting, and begs for a simpler, tractable, and computationally enforced scheme that can scale to large systems composed of many components. In such a scheme, OSS/CSS licenses could formalize IP obligations as operational requirements (i.e., computationally enforceable) at the integrated system level and instantiated by system integration architects. Similarly, customer/user rights are then nonfunctional requirements that can be realized and validated as access/update capabilities propagated across the integrated system. [3]

Cybersecurity for OA systems

Cybersecurity is a high priority requirement in all C2 systems, applications, and platforms. [14] No longer is cybersecurity something to be addressed after C2 are developed and deployed — cybersecurity must be considered throughout the design, development, deployment and evolution of C2. However, the best ways and means for addressing cybersecurity requirements are unclear, and oftentimes somewhat at odds with one another depending on whether cybersecurity capability designs are specific to the: C2 platform (e.g., operating system or processor virtualization; utilization of low-level operating system access control or capability mechanisms); component producer (secure programming practices and verification testing); system integrator (e.g., via the use of secure data communications protocols and data encryption); customer deployment setting (mobile: airborne or ship-board; fixed: offices, briefing rooms, operations centers); end-user authentication mechanisms; or acquisition policy (e.g., reliance on third-party audit, certification, assurance of system cybersecurity). However, in reviewing these different arenas of cybersecurity, we have found that the cybersecurity requirements or capabilities can be expressed in much the same way as IP licenses: using concise, testable formal expressions of obligations and rights. Some suggestive examples follow (capital letters are placeholders that denote specified system, service, or component contexts).

- The obligation for a user to verify her authority to invoke application software and access data in compartment T, by password check or other specified authentication process.
- The obligation for all components connected to specified component C to grant it the capability to read and update data in compartment T.
- The obligation to reconfigure a system in response to detected threats, when given the right to select and include different component versions, or executable component variants.
- The right to read and update data in compartment T using the licensed component C.
- The right to replace component C with a certified secure component D.

These examples show how cybersecurity requirements can be expressed or paraphrased into/from restricted natural language (e.g. using a domain-specific language) into composite specifications that denote “security licenses.” [1, 2, 14] In this way, it should be possible to develop new software analysis tools whose purpose is to interpret cybersecurity obligations as operational constraints (executable) or provided capabilities (access control or update privileges) through mechanisms analogous to those used for analyzing software licenses [1, 4, 14], and to determine how component or subsystem-specific obligations and rights can be propagated across a system’s architecture. Consequently, we believe that cybersecurity can therefore in the future be addressed using explicit, computational OA representations that are attributed with both IP and cybersecurity obligations and rights.

Build, Release and Deployment (BRD) Processes and Process Automation

C2 applications represent complex software systems that are often challenging to produce and maintain, especially when initially conceived as bespoke systems. To no surprise, acquisition of these systems requires a development life cycle approach, though some system elements may be fully formed components that are operational as packaged software (e.g., commercial database management systems, web browsers, web servers, user interface development kits/frameworks). C2 system development is infrequently clean-sheet and even less likely to be so in the future. As a result, component-based system development approaches are expected to dominate. This implies system integrators (or even end users) must perform any residual source code development, inter-app integration scripting, or intra-app extension script development. But software process challenges arise along the way. [13]

First, as noted earlier, the issue of whether there is an explicit, open source OA design representation — preferably one that is not just a diagram but is instead expressed in a computational architectural design language. With only a diagram or less, there is little or no guidance for how to determine whether/how an op-
eral software implementation is verifiable or compliant with its OA requirements or acquisition policies. Current acquisition policy guidance calls for provision or utilization of standardized, open APIs intended to increase software reuse, selection of components from alternative producers, and post-deployment system extensions. [6]

Second, there is the issue arising from system development practices based on utilization of software components, integrated subsystems, or turn-key application packages. These software elements come with their own, possibly unknown requirements that are nonetheless believed to exist and be knowable with additional effort. [3] They also come with either OSS code or CSS executables, along with their respective APIs. These components must be configured to align with the OA specification. Consequently, software tool chains or workflow automation pipelines are utilized to build and package internal/external executable, version-controlled operational software releases. We have found that many diverse automated software process pipelines are used across and sometimes within software integration activities. [13] These pipelines take in OSS code files, dependent libraries, or repositories (e.g., GitHub) and build executable version instances that are then subjected to automated testing regimes ranging from simple “smoke tests” to extensive regression testing. Successful builds eventually turn into packaged releases that may or may not be externally distributed and deployed as ready-to-install executables. While this all seems modest and tractable, when one sees the dozens of different OSS tools used in different combinations across different target platforms, it becomes clear that what is simple in the small becomes a complex operations and maintenance activity when the scale of deployment increases.

Another complication that is now beginning to be recognized within and across BRD processes and process automation pipelines arises in determining when and how different BRD tool chain versions/configurations can mediate cybersecurity requirements in the target system being built or maintained. For instance, many software system builds and deployed releases are assumed to integrate to functionally equivalent CSS components. However, many CSS components are not included in distribution releases due to IP restrictions and thus must be linked and configured into already deployed operational systems. We have also observed and reported how functionally equivalent variants as well as functionally similar versions may or may not be produced by BRD tool chains, either by choice or by unintentional consequence. This, in our opinion, gives rise to the need for explicit open source models of BRD process automation pipelines that can be analyzed, reused, and shared, as well as systematically tested to determine whether release versions/variants can be verified and/or validated to produce equivalent or similar releases that preserve prior cybersecurity obligations and usage rights.

Software Evolution Practices Transmitted Across the OA Ecosystem

Software evolution is among the most-studied of SE processes. While often labeled as “software maintenance,” a frequently profitable activity mediated through maintenance contracts from software producers to customers, the world of OSS development projects and practices suggest a transition to a world of continuous software development — one that foreshadows the emergence of continuous SE processes or software life cycles that just keep cycling until interest in the software falters or spins off into other projects. OSS development projects rely on OSS tools that themselves are subject to ongoing development, improvement, and extension, as are the software platforms, libraries, code-sharing repositories, and end-user applications utilized by OSS developers to support their development work. The migration of developers within/across OSS projects further diversifies the continuous development of the most successful and widely used OSS components/apps. This dynamism in turn produces many ways for OSS systems, or OA systems that incorporate OSS components, to evolve.

Figure 2 portrays different software evolution patterns, paths, and practices we have observed arising with new C2 applications. [12] Here we see paths from a currently deployed, operational software system release to a new deployed release update — something most of us now accept as routine, as software updates are propagated across the Internet from producers, through integrators, to customers and end users.

Integrated OA systems can evolve through upgrades of functionally equivalent component variants (patches) as well as through substitution of functionally similar software components sourced from other producers or integrators. In Figure 3, we show a generic situation that entails identifying how an OA consistent with that depicted in Figure 1 may accommodate the substitution and replacement of a locally installed word processor application program (like Microsoft Word) by a remote web-based word processing software service (for example, Google Docs or Microsoft Office 365). Also note how the software IP licenses compose across the four similar release instances shown.

This capability is a result of utilizing an OA that constitutes a reference model aligned with a vendor-neutral software product.

Figure 2. Different paths and mechanisms through which currently operational OA software system components can evolve. [12]
line. This is also a capability sought by customer organizations, and sometimes encouraged by software producers to accommodate their evolving business models (discussed below). While the OA remains constant, the location of the component has moved from local to remote/virtual, as has its evolutionary path. Similarly, the propagation of IP and cybersecurity requirements for the local versus remote component has changed in ways that are unclear and entail a different, evolved assurance scheme.

Overall, the evolutions of software components, component licenses, component interconnects and interconnected component configurations are now issues that call for SE research efforts to help make such patterns, paths, and practices more transparent, tractable, manageable, and scalable within an OA software ecosystem, as well as for customer organizations that seek the benefits of openness, sharing and reuse.

**New Business Models for Ongoing Acquisition of Software Components and Apps**

The last issue we address is the newest in this set of six for consideration for new SE research. While SE research and practice has long paid attention to software economics, the challenges of software cost estimation are evolving in light of new business models being put into practice by software producers and system integrators.

In the past, system development projects were often managed by a single contractor responsible for both software production and system integration. Costs could be assessed through augmentation of internal business accounting practices (e.g., budgeting, staffing workloads, time sheet reports, project schedules, fixed fees, etc.). But a move to OA ecosystems means that multiple producers can participate, and OA schemes accommodate switching among providers while a system is being integrated, deployed, or evolved in the field. This in turn coincides with new ways and means to electronically distribute software updates, components, or applications as well as new ways to charge for software. For example, OSS components may be acquired and distributed at “no cost” but their integration and evolution charged as a service subscription or as time-effort billings.

We have already seen other alternatives for costing, paying for, or charging for software that include franchising; free component, paid service/support fees; enterprise licensing; metered usage; advertising supported; subscription; federated reciprocity for shared development; collaborative buying; donation; sponsorship; government-provided free/open source software (e.g., Government OSS – GOSS); and others. So how are customer organizations, especially in the Defense community where software cost estimation practices are routine, suppose to estimate the development or sustaining costs of the software components or integrated systems they acquire and evolve? How are software costs to be estimated when an OA system allows for producers whose components come with different costing/billing schemes? This is an open problem for SE research in industry practice.

The last piece of the puzzle we are studying is the envisioned transition within the Defense community to C2 systems being composed by customer organizations, and possibly extended by end users deployed in the field. This is the concept that surrounds the transition to discovering and acquiring software components, apps, or widgets in Defense community app stores. [6] These app stores are modeled after those popularized for use in distributing and acquiring software apps for web-based or mobile devices, like those operated by Apple, Google, Microsoft, and others. How the availability of such Defense community app stores will transform the way C2 systems are produced and maintained — or even whether they will be produced by legacy Defense industry contractors — remains to be seen. Said differently, how app stores transform OA software ecosystem networks, business models, or cybersecurity practices is an emerging challenge for SE research.

**Discussion and Conclusions**

In this paper, we have focused attention on software engineering research challenges that are emerging in the Defense community. Much of the earlier research and advances in SE emerged from challenges in this same community 40 to 50 years ago. Most contemporary SE research has moved away from this community. However, as we sought to describe in this paper, this community is again surfacing and facing a growing myriad of issues and challenges that can directly benefit from targeted advances in SE research and practice.

We identified and examined six target areas for SE research that now plague the Defense community, and perhaps other industries and other government agencies as well. All of these SE research areas are readily approachable, and research results are likely to have significant practical value, both within the Defense community and beyond.

These issue areas were investigated and addressed in the domain of command and control systems (C2). We believe all
these issues are tractable, yet dense and sufficient for both deep, sustained research study and also for applied research in search of near-term to mid-term practical results.

In related work [15], we call for specific R&D investments into the development of open source, domain-specific languages for specifying open architecture representations (or architectural description languages, “ADLs”) that are formalizable and computational, and for supporting annotations for software license obligations and rights. While ADLs have been explored in the SE research community, the challenges of how software architectures mediate software component licenses and cybersecurity requirements is an open issue with practical consequence. Similarly, ADL annotations that incorporate IP and cybersecurity requirements, along with costs or cost models in line with new software business models, is an open problem area. We have also called for R&D investment in new SE tools or support environments whose purpose is to provide automated analysis and support of OA systems’ IP and cybersecurity obligations and rights as new requirements for large-scale software acquisition, design, development, deployment and evolution. Such environments are the automated tools that could be used to model, specify, and analyze dynamically configurable, component-based OA software systems expressed using the open source architectural representation schemes, or ADLs, noted here. We hope this paper serves to help throw light into these otherwise dark corners of SE research that can inform and add benefit to software development practices for C2 and enterprise business systems for use and evolution throughout the Defense community.

Acknowledgments

The research described in this report was supported by grants N00244-15-1-0010 and N00244-16-1-0004 from the Acquisition Research Program at the Naval Postgraduate School, Monterey, CA. No endorsement, review or approval implied. The concepts, topics and materials presented are the sole responsibility of the authors.

REFERENCES


ABOUT THE AUTHORS

Walt Scacchi is senior research scientist and research faculty member at the Institute for Software Research, University of California, Irvine. He received a Ph.D. in information and computer science from UC Irvine in 1981. From 1981 to 1998, he was on the faculty at the University of Southern California. In 1999, he joined the Institute for Software Research at UC Irvine. He has published more than 200 research papers and has directed more than 65 externally funded research projects. In 2011, he served as co-chair for the 33rd International Conference on Software Engineering—Practice Track, and in 2012, he served as general co-chair of the 8th IFIP International Conference on Open Source Systems.

Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA
wscacchi@ics.uci.edu

Thomas A. Alspaugh is a project scientist at the Institute for Software Research, University of California, Irvine. His research interests are in software engineering, requirements, and licensing. Before completing his Ph.D., he worked as a software developer, team lead, and manager in industry and as a computer scientist at the Naval Research Laboratory on the Software Cost Reduction (or “A-7”) project.

Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA

**John D. Rockefeller and Alexander Hamilton**

**The Founding Fathers of Agile**

*Peter D. Morris, PMP, PMI-ACP*

“*My country has contrived for me the most insignificant office that ever the invention of man contrived or his imagination conceived.*”

— Vice President, John Adams

---

**Introduction**

Nothing has ever been invented — not the light bulb, not the automobile, not the iPhone. Someone at some point got the notion to combine two or more existing ideas and call it the next new thing.

So it was with the Agile Alliance, the Agile Manifesto and its nine principles. In February 2001, 17 “anarchists” met to create a common and more efficient method of software development. They determined what was valuable, productive, essential, negotiable, deliverable, trustworthy, sustainable, creative and simple. [1] Eight of the “anarchists” then posed for that famous brown tinted faded photo as though they had just drafted the Declaration of Independence. But as with all inventions, their work was based on the ideas of at least two famous individuals at least a century or two earlier.

In the summer of 1855, 16-year-old John D. Rockefeller, the itinerant son of a snake oil salesman, unable to afford a formal college education, completed a three-month course of study at E. G. Folsom’s Commercial College, a chain trade school with branches in seven cities. He acquired working knowledge in double-entry bookkeeping, penmanship, and the essentials of banking, exchange and commercial law. By January 1857 he had been elevated to chief bookkeeper for a Cleveland merchant, mainly because his limited business training allowed him to determine whether his company was gaining or losing money and to enact process improvements to make more money (unlike his competitors, who generally had no idea they were losing money until they went out of business).

Shortly thereafter he started “Clark and Rockefeller,” which bought and sold carloads of produce. For him, the Civil War was principally an opportunity to pile up riches. Once oil was struck in western Pennsylvania, his company took on consignment some of the first crude oil shipments, which reached his Cleveland refineries in early 1860. Known today erroneously as an oil baron, John D. was known as the “Kerosene King,” owning multiple refineries and controlling the trains and pipelines that carried the illuminant around the world. It would be 50 years before John D. found a useful purpose for gasoline, an unwanted byproduct of the oil refinery process. [2]

John D. was one of the first to apply the nine Agile principles. Before he began distributing kerosene, only rich people could light their houses after dark (candles cost money). He satisfied customers through early and constant kerosene delivery (OK — he broke a few laws making deals with the railroads, but that forced him to satisfy the Agile concept of reducing documentation). He welcomed changing requirements by going from produce to oil and shortened timescale by making (mostly illegal) deals with suppliers face to face. He had to work with the railroad and pipeline suppliers, motivating their self-organizing teams through greed. Whenever government regulators questioned his business acumen, he would reflect, tune and adjust his behavior to satisfy their concerns.

A century earlier, Alexander Hamilton also began as a clerk at 16. He worked for the firm Beckman and Kruger in St. Croix. However, a grim tally of disasters preceded this. Between 1765 and 1769, his father vanished, his mother died, his cousin and supposed protector committed bloody suicide, and his aunt, uncle and grandmother all died. Alexander was left alone, friendless and penniless, surrounded by failed, broken, embittered people. His short life had been shadowed by a stupefying sequence of bankruptcies, marital separations, deaths, scandals and disinheritance. Such repeated shocks must have stripped Hamilton of any sense that life was fair, that he existed in a benign universe, or that he could ever count on help from anyone. But help finally arrived in the form of benefactors who sent him to America. There he attended college for a time, then became George Washington’s adjutant during the Revolutionary War. After that, he became Secretary of the Treasury under Washington’s presidency. [3] Hamilton worked together in the field with Washington, coordinating self-organizing commands through face-to-face communications, motivating soldiers to work together to maintain a constant pace of battlefield victories on a short timescale. When faced with adversity, he and Washington reflected, tuned, and adapted their behavior until their eventual victory at Yorktown. As Treasury Secretary, Hamilton satisfied his mercantile customers with early delivery of the First National Bank (to the chagrin of his agrarian enemies Thomas Jefferson, James Madison and James Monroe, proving you can’t please everybody). When his plan to fund the government through customs duties was threatened by smugglers, he quickly changed requirements and created a customs enforcement navy that eventually became the Coast Guard.

**So You Want to be an Agile Project Manager**

A lot has changed in the past couple centuries, but those basic Agile concepts are still valid. A typical Agile team consists of a product owner (PO), Scrum master (SCM), and team members...
who perform multiple tasks (defining scope, establishing requirements, designing, coding and testing). Notice I didn’t list project manager (PM), since most Agile teams do not have one. Hamilton and Rockefeller would likely fall into the PM/PO category. In a recent unscientific blog post[4] that I posted (there is very little research in this area), I received 72 comments on this subject from both Agile PMs and Agile practitioners. Most PMs thought that their skills could benefit Agile teams. However, most Agile practitioners not only viewed the PM as evil but also considered all project managers to be wastes, bottlenecks and a hindrance to velocity. Many questioned whether a PM should be allowed to breathe oxygen in an Agile environment. The typical PM is considered to be like a wart on a frog: defined in the SOW as necessary (a holdover from Waterfall), but no one is quite sure what he or she does. I’ve personally met several Agile PMs who admitted they really had very little to do. I believe this is because while there are many books, tutorials and classes that teach project management and Agile methodology, no one has ever attempted to define Agile PMs in a way that warrants respect and usefulness.

So let’s try to define some major life cycle processes — and I’ve purposely chosen some that make most Agile practitioners cringe — where the PM can demonstrate effectiveness: requirements management, Agile leadership, configuration management and change control, team composition and management, scope and estimation, monitoring and control, measurement and analysis, Agile product integration, Agile release planning, and prioritization and tracking. Because many Agile PMs are their own worst enemy, as Hamilton and Rockefeller often were, failing to understand their basic role in an Agile environment, let’s start with requirements management and leadership.

Requirements Management

“And you want to be my latex salesman?”

—Jerry Seinfeld, “The Boyfriend” episode

Since Scrum is the most popular of the Agile frameworks, this article will target process challenges for that method. It should be noted that multiple frameworks can be combined (e.g., Kanban can be used within Scrum to control team capacity constraints by limiting work in progress). While the goal for management is to make sure the product/iteration backlog gets fed and can support and describe what needs to be built by the development team prior to the start of the sprint, many project managers and organizations struggle to understand value-based decomposition and backlog prioritization. [5]

Hamilton spent a great deal of time convincing the states to ratify the Constitution over the existing Articles of Confederation. The first would create a powerful centralized federal government while the latter provided more states’ rights, so the South figured, “Why fix what ain’t broke?” As Hamilton negotiated with each state, requirements changed (Virginia demanded slavery; New York was abolitionist; South Carolina was agrarian; the North mercantile). Similarly, management of Agile requirements differs from traditional requirements management because requirements evolve with iterations. With Agile methods, the team does not attempt to specify fully detailed requirements up front. Instead, the team initially establishes “course grained” requirements and then progressively elaborates on them as the process progresses. The requirements are refined from the product backlog into the iteration goal, then further refined into the iteration plan and then into user stories. The team continues to refine the user story requirements during the daily stand-up meetings and through other discussions. These are practical examples of how project elements get decomposed and prioritized at the last responsible moment. The term “minimally marketable feature” (MMF) is an important factor in a successful iteration in order to avoid what is termed “gold plating” in traditional development environments. MMF refers to a package of functionality that is complete enough to be useful yet too small to be representative of the entire project. [6] MMF also allows for the immediate delivery of basic working product to the end user with the option to add features later … or not. Rockefeller’s MMF was kerosene; he could have controlled the oil wells, pipelines and rails (which he would do much later), but for at least the first few decades he concentrated on what he could deliver quickly and efficiently. Hamilton’s MMF was the preservation of the Union under the Constitution, and he quickly curbed his abolitionist tendencies so that the South would support both.

Agile projects are subject to additional constraints not normally associated with development projects. These constraints mainly involve tasks that fall outside the scope of the project work, including:

—How to effectively manage an organization’s internal personnel so that the appropriate stakeholders are available throughout the course of a project.
—How to gather and prioritize the most important features desired by the organization throughout the ongoing development cycle.
—How to adjust the notion of needed training in a continual release environment.
—How to ensure customer team members are informed by the full breadth of stakeholders required for enterprise acceptance.
—How to secure timely approval of new technologies that a team would like to leverage.
—How to address stakeholder discomfort with cultural, business, social or other nontechnical changes related to Agile software implementation.

Project teams are encouraged to consider these issues, as well as mitigation strategies employed to resolve the associated risks, when considering Agile requirement elicitation policies and procedures.

Honesty, Integrity and Agile Servant Leadership

“If your actions inspire others to dream more, learn more, do more and become more, you are a leader.”

—John Quincy Adams

In their book “The Wisdom of Teams,” [7] Jon Katzenback and Douglas Smith define a real team as “a small number of people with complementary skills who are committed to a com-
mon purpose, performance goals and approach for which they hold themselves mutually accountable." This and other research on the subject defines a high-performance team as one that exhibits all these traits plus self-organization and direction, traits typically exhibited by mature Agile teams.

Members of such empowered teams are freed from command-and-control management and can use their own knowledge to determine how to best do their job. Hamilton knew we needed a central bank and entrusted its foundation to people who understood banking on the European model. Rockefeller understood the need to turn murky oil dredged from the Pennsylvania fields into kerosene but left the creation and operation of the refineries to those teams that understood it. Empowering teams also enables organizations to tap into people’s natural abilities to manage complexity, thus reducing or eliminating the need for teaching and coaching individuals. Instead of providing detailed task lists, leaders of Agile teams should describe the iteration goals at a high level and let the team determine how best to accomplish the work. This self-organization naturally leads to self-directing teams that work together to create team norms and to make their own local decisions (i.e., at daily stand-ups).

This recognition that the team is in the best position to organize project work is liberating and motivating for team members. This practice alleviates many of the technical blockages seen in push systems where the task list and sequence is imposed on the team. Agile promotes a servant leadership model that recognizes that it is the team members, not the project manager, leader, coach or Scrum master who get the technical work done and achieve business value.

Many Agile project managers struggle to comprehend their role as servant leader. I’ve spoken with a number of PMs who basically told me that it wasn’t their job to ‘carry water’ for the team. My advice to these prima donnas was to suck it up, forget about hierarchy and become a member (rather than the dictator) of their team. The servant leadership management approach redefines the leader’s role in relation to the team. It focuses the leader on providing what team members need, removing impediments to progress, and performing supporting tasks to maximize the team’s productivity. This makes sense in a technical environment where the product owner grooms and prioritizes the backlog and the team meets daily to resolve risks and issues—all tasks that would normally fall under the domain of the project manager in a traditional environment.

There are four primary duties a leader performs in this role as servant to the team:
1. Shield the team from interruptions.
2. Remove impediments to progress.
3. Communicate the product vision.
4. Provide the essential resources (tools, compensation, encouragement) team members need to keep them content and productive.

In addition to these four core duties, there are other activities that servant leaders should keep in mind: [8]
1. Learn the team members’ needs.
2. Learn the project’s requirements.

3. Act for the simultaneous welfare of the team and the project.
5. Have a vision of the completed product (understand the definition of “done”).
6. Use the project vision to drive your own behavior.
7. Serve as the central figure in successful project team development.
8. Recognize team conflict as a positive step.
9. Manage with an eye toward ethics.
10. Remember that ethics is not an afterthought but is an integral part of thinking.
11. Take time to reflect on the project.
12. Develop the trick of “thinking backward.”

“Thinking backward” means visualizing the end goal and then working backward to determine what needs to happen to get there and what problems and risks may arise. It may seem strange to limit a discussion of requirements development and management to decomposition, prioritization and servant leadership. Remember, however, that most development problems begin with requirements, and how they are broken down into their component parts, prioritized for value, and managed can mean the difference between high quality, increased velocity and stakeholder dissatisfaction.

The Agile team must be trained at project inception and coached by a knowledgeable advocate who can help guide them through the process. It is important that the Agile PM sets aside funding during the early acquisition stages for initial and ongoing training and support. Team formation typically follows the Adaptive Leadership [9] stages of forming, storming, norming and performing associated with the project management activities directing, coaching, supporting and delegating. Not all teams go through these phases in a predictable way, because each team is different. The best an Agile PM can do is to be aware of these models, look for signs that the team is in a particular phase, and then lead accordingly. While such general pointers are useful, the Agile PM should never expect teams to proceed in an orderly fashion and follow the traditional stages.

Rockefeller well understood the concepts of high-performance teams and servant leadership. It’s very hard to establish the world’s largest trust without such characteristics. He got in trouble for it with two presidents, but more on that later. Hamilton, though, often adopted a “my way or the highway” approach, drowning his critics in words and paper. It was only through the intervention of Washington that Hamilton lasted more than a day in office.

Configuration Management and Change Control
“The guy who knows about computers is the last person you want to have creating documentation for people who don’t understand computers.”

—Adam Osborne

The typical Agile practitioner will refer you to the Agile Manifesto (it’s their Bible) whenever possible, especially regarding this aspect of project management. They’ll quote “working software over comprehensive documentation.” They really, really hate document-
tation, arguing that code comments are sufficient or that, due to the simplicity and inherent structure of modern code, there is no need for separate documentation. As an Agile PM, some of your greatest feats will be convincing Agile practitioners to document anything outside of code at all and convincing them that ‘minimal documentation’ is not ‘maximum invisible documentation.’ In a traditional development methodology, change control meetings are used to review and manage change. This is needed because the project is long enough that it has passed a requirements baseline phase and changes must now be managed. Because Agile works in short iterations, change control is not needed and becomes embedded in the iteration planning session. Of course, changes can be identified at any time and should be captured and documented. However, the changes should be discussed during the next iteration planning session and not during an iteration of work. There are times that a change stops the iteration due to its nature, but this is rare.

**Team Composition and Management**

“Individual commitment to a group effort: that is what makes a team work, a company work, a society work, a civilization work.”

—Vince Lombardi

Team composition on Agile projects needs to reflect throughput maximization. You staff the team to be able to deliver the most potentially deployable quality software at the conclusion of every iteration/sprint with as little waste as possible. The ideal combination of team composition and management involves such factors as advocacy, communications, consistency and continuity. Small teams are preferred; in fact, at the beginning of John Adams’ first term in office, people were astounded that the federal government had ballooned to 35 personnel, including Adams and five cabinet members.

A 1995 study [10] describing how merchant marines bring their ships into port used the term “distributed cognition” to describe how the crew operates as though using a common brain with distributed components. This phrase helps us understand why proximity and collaboration are so important on Agile software projects.[11] Each person on the team is busy forming a slightly different idea of what problem he or she is solving and how the solution should look. Each is encountering problems that someone else on the team might be able to help with.

Viewed in this way, we see software development as a problem of mental search and synchronization. We add the difficulty of resolving differing opinions and learning how other people work, what motivates them, what angers them, and so on. We see, from this perspective, why communication and collaboration are so important on software projects, and especially within short Agile timeboxes.

The Allen curve [10] is a graphical representation that reveals the exponential drop in frequency of communication between engineers as the distance between them increases. During the late 1970s, Thomas J. Allen undertook a project to determine how the distance between engineers’ offices affects the frequency of technical communication between them. That research produced what is now known as the Allen Curve. The research indicated a strong negative correlation between physical distance and the frequency of communication between work stations. The study also reported that communication drops off at about 10 meters in distance — basically, people won’t walk farther than the length of a school bus to ask a question. For this reason, close proximity and co-location are essential for Agile team success whenever possible.

Another challenge is keeping high-performing Agile teams together long enough for them to achieve peak performance. For this reason, Agile teams do not flourish in a matrix environment, where developers and testers are pooled and report to a separate entity. However, while Agile teams do well in projectized environments where the entire team reports to a common entity (i.e., the project manager), the developers can still change at the end of a contractual period of performance. The continuity of an Agile team enhances the “tacit knowledge” of the program — knowledge that is not written down but is instead supported through collective group knowledge, which tends to improve overall performance. It is therefore essential that every effort be made to either keep a high-performance team working together or, at a minimum, ensure that a core constituency remains even when other personnel are on the move.

Hamilton and Rockefeller both understood the need for co-location. Rockefeller based his trust in Cleveland (near the oil fields) and later New York (near the money). Hamilton’s greatest concern and victory was for a centralized government rather than a loose confederation of states as proposed by Jefferson and Monroe. He understood early on that the country couldn’t long exist without central executive, legislative and judiciary branches of government to handle money, taxes, war and the general good on a grand scale. No large country in history had succeeded in democracy without sliding into dictatorship, as France would prove in 1804. He then managed the deciding vote to move the capital from Wall Street to Washington, D.C., by promising the Pennsylvania congressional representatives a temporary 10-year capital in Philadelphia (who assumed they’d never build a capital out of that swampland next to the Potomac), thus exhibiting the traits of collaboration over contract negotiation. He also considered individuals and interactions by placing the planned capital next to the Potomac near Mount Vernon, thus gaining the support of George Washington.

**Scope and Estimation: Planning Poker**

“Informing decision making comes from a long tradition of guessing and then blaming others for inadequate results.”

—Scott Adams

Probably the greatest concern for both Rockefeller and Hamilton was how to manage that huge scope of money and power, respectively. Rockefeller did it through double-entry accounting — it was said that even in his seventies he could quickly scan a lengthy ledger and spot a one-penny mistake. Hamilton managed the scope of his endeavors through lengthy reports and three-hour speeches. (OK, he wasn’t completely Agile, and he was a lawyer.) Similarly, one of the biggest questions a PM
will get from Agile practitioners is why they should estimate the current iteration at all. They see it as a waste of time since (a) they will eventually break down all their tasks into one-hour increments, and (b) anything they don’t finish in this iteration will be finished in the next. This is fine if you’re a code dork who communicates with little more than a screen and is responsible only for churning out code; it’s not so fine for a PM who has to communicate with the sponsor and explain what is complete, what is yet to be complete, and what will likely not be addressed.

There are many estimation techniques (i.e., Wideband Delphi, White Elephant, Affinity) that work just fine for short Agile iterations. But given the aversion of most teams to estimation at all, Planning Poker affords the most involvement and, therefore, accountability. Planning Poker is a consensus-based game where team members individually estimate user stories using numbered cards. This is followed by iterative discussions until consensus is reached. A variation of the Wideband Delphi technique, team members play their estimate cards face down prior to any discussion. Planning poker minimizes anchoring—the tendency of strong-willed individual voters who may have specific agendas that exert undue influence on the team (we’ll call it the Jefferson effect).

Story estimates may be done in actual days, ideal days or story points. All team members have identical card decks containing the Fibonacci sequence (1, 2, 3, 5, 8, 13, 21, 34, 55, etc.). Story points are preferred and are used to estimate larger pieces of work (i.e., user stories and epics). If your cards include Fibonacci numbers over 21, it’s time to break down your tasks to more manageable chunks.

The process of the Planning Poker session is as follows:

- The meeting is facilitated by a Scrum master from another team who does not play.
- For each story, the team asks the Scrum master questions and discusses assumptions, constraints and risks.
- To vote, each team member plays a card face down that represents his story estimate. This is followed by discussion, during which no mention of the unrevealed story estimate is made. Then all players simultaneously reveal their cards by turning them over.
- Outliers (i.e., the players with highest and lowest estimates) are given an opportunity to explain their estimates to the team.
- The process continues in an iterative fashion until consensus is reached for each story.
- If stories are deemed too large to be included in the iteration, they may be divided or deferred to a later iteration.

**Monitoring and Control**

I was a student of Frederick Taylor’s “Time and Motion Studies” in college. One day, the professor was teaching us how to time a worker’s motions using a stopwatch: reach, grab, lift, return, drop. He told us, “Gentlemen, make no mistake of it. While you’re timing that worker’s motions, he hates your guts.” I mention this because Agile practitioners generally point to the Agile Manifesto or Principles (do they memorize these or carry them around in their pockets?) to argue against the need for metrics and monitoring. They’ll argue for “working software as the primary measure of progress,” the need for simplicity, welcoming change and maintaining a constant pace (for which metrics would be a roadblock). This is fine for a product owner who understands how to read a burn down chart and maintains constant vigilance on the product backlog. Unfortunately, that PO has a boss who has a boss, neither of whom understand anything but numbers. This is why senior management loves EVMS (i.e., SPI/CPI = 1.0 good, anything else is probably bad). While seemingly simple, explaining a burn down chart to a senior manager can be like trying to explain alternate side of the street parking to a cranberry.

Agile methods place a greater emphasis on products delivered than on attributes of the process used to create those products. Measuring durations, costs and schedule performance—while not unimportant—tends to receive less attention in Agile approaches. Counts of delivered story points (embodied in working software) are the most prominent building blocks in Agile metrics as a byproduct of coding. One of the greatest benefits of Agile, due to its iterative nature, is its ability to allow for changes by the sponsor not only between but also during the short iterative cycles. Since Agile produces working software every 30 to 90 days on average (and in many cases, every one to two weeks), it could easily be argued that all Agile projects operate in the maintenance and sustainment phase of the software life cycle, where user demands trump any requirements agreed upon in the initial acquisition. This is especially important where safety of life (or, for financial clients, safety of money) is involved, where the inability of projects to turn on a dime can have catastrophic results in fielded software.

Consequently, of particular concern is the reporting and milestone requirements typically levied against Agile contracts, which may be especially difficult to achieve using Agile methods. As an example, consider an Agile team entering a preliminary design review (PDR) where the main concern is to ensure that the sponsor review team understands what is and is not included in the PDR. Requirements assigned to future iterations would have no detailed designs available, whereas in a traditional PDR all designs are typically completed. The selection and implementation of appropriate contracting vehicles to support the types of practices that successful Agile projects exhibit is a particular management concern. It is therefore important to concentrate on those Agile challenges and novel approaches when the traditional milestone review collides with Agile development.

**Definition of Done**

This is where that elusive concept “definition of done” needs to be established to the satisfaction of both the team and the sponsor in support of progress reporting. An Agile story is done when the function of that story is demonstrated to the customer and the customer agrees that the functionality substantially fulfills what is expected. [12] Given the incremental nature of Agile work, the most productive approach to technical reviews will also be incremental in nature, although an incremental approach to reviews may not always be possible. One popular approach to incremental reviews in Agile development is test-driven development (TDD). [13] In this approach, rather than developing tests to verify an already built code base,
test cases are developed first (and often automated) and the code is written to pass these tests. In this scenario, technical progress can be understood through metrics that show the pass/fail outcomes of the tests. In fact, for most Agile projects the definition of done is the successful passing of established acceptance tests. For many Agile projects the simplest and most accurate representation of progress is “story points promised vs. story points delivered” (which roughly translates on a traditional project to “estimates vs. actuals”), since delivered features are a composite of story points.

Earned value is one of the primary tools that the Department of Defense uses to measure contractor performance. For programs valued at more than $20 million, an earned value management system (EVMS) is required to be used, and for programs more than $50 million, a validated EVMS must be used. “EVM techniques, however, assume complete planning of a project to discrete work package levels, then assigning cost and duration to these packages.” [14]

AgileEVM calculations are based upon delivery of completed, tested units of functionality. No credit is given for delivery of intermediate work products. Therefore, while AgileEVM may be seen as incorporating quality standards into the metric and may be seen as providing stricter evidence with respect to delivery of value, proponents of traditional EVMS may need to be consulted on the difference between performance indexes that measure product progress (traditional) and value enhancement (AgileEVM).

Measurement and Analysis

Start by specifying measures. At their core, Agile methods are “inspect and adapt” methods. Inspection implies measurement. However, there is very little explicit Agile industry guidance on how and what to measure. While each Agile PM must determine metrics applicable to their practices, such metrics should be selected to provide transparency, improve effectiveness, and enable continual improvements. Most importantly, the PM should consider the metric contribution to the value stream. If it doesn’t directly or indirectly provide value to the customer, collection and analysis of the metric should be challenged. Potential measurements for an Agile IPT may include the following:

- **Burn down**: Keeps track of remaining effort; encourages early course correction.
- **Velocity**: Amount of product backlog a team can complete within a given iteration; provides a basis for planning and estimating (in some cases “story point” collected and completed can be used to determine iteration velocity). Remember that velocity between teams is not comparative or additive, since it does not show relative performance or productivity of teams.
- **Churn**: This metric shows how many lines of code the team added, deleted or modified during the most recent iteration. With the code churn metric, you can analyze how source files are changing over time.

Each Agile PM should spend time determining which metrics are most applicable to their unique practices. Metrics should be selected not only for transparency, effectiveness, and enabling continual improvements, but also with regard to the strategic goals of the organization and the project itself. In addition to analyses conducted at the review and retrospective meetings, Agile teams typically perform metric analysis at the daily team meetings where iteration and release burn down data are analyzed.

Velocity constitutes one of the more useful Agile analyses. Solid Agile teams have a consistent velocity (+/- 20 percent). Fluctuations in the velocity metric may indicate a need to stabilize a team or environment in terms of training or tool usage. It’s important to note that velocity is not to be used as comparison between teams, since it does not show relative performance or productivity of teams. Velocity is a measure of the number of features developed by an individual or team during an iteration. It does not account for size, and therefore comparing a velocity of 20 one-hour features with one of six eight-hour features can be misleading at best.

**Agile Product Integration**

“There are only three things that can happen when you pass the ball, and two of them are bad.”

—Paul “Bear” Bryant on risk management

The Agile approach toward integration should be to do it continually, and therefore documentation should be done continually throughout the life cycle. The general Agile rule is if every developer integrates and commits every day to the baseline, a big system could be developed by slicing the products in such a way that many integration points are planned. Therefore documentation should occur once a week or once a month, depending on the project. Of course this creates overhead, but risk is also sliced and minimized throughout the project. Remember that Agile favors working product over lengthy documentation, not instead of. This is a common myth that can lead to project failure when documentation is insufficient to properly describe integration planning.

Agile integration procedures and criteria should ideally be generated automatically if they are to be useful in a fast-paced development environment. One potential artifact that might satisfy these criteria is the “Agile functional specification,” especially in the form of user stories that evolve in conjunction with the client or their business representative. This implies that as the user stories evolve, the technical specifications will evolve as an automatic derivative. To achieve this, the user stories must be written in a context-aware structured syntax — in other words, a domain-specific language (DSL). Then it will be possible to consume the user stories and, from them, auto-generate the interfaces and the unit tests required to create an Agile technical specification. The interfaces are derived from the user story setups and the unit tests from the user stories.

If Agile is to succeed, testing (e.g., TDD) must become a central pillar of the development process. Incorporating testing at the end of a major project will only lead to show-stopping bugs and major timeline setbacks. To ensure that the product is of high quality, Agile PMs should include test personnel from the very beginning so they can spot issues early on and prevent major problems down the line. This is a major component of Agile, and process infrastructure should be in place to record test results throughout the product life cycle.
The phrase “fail early” is becoming very popular in the Agile world. In essence, it means “figure out as early as possible whether or not what you are doing will succeed.” As simple and as obvious as this may sound, it is amazing how often critical, high-risk items are left until the final stages of the project. This is one of the biggest reasons for project failure. Handle the risky stuff first and “fail early.” Hamilton began creating a Central Bank early in his tenure as Treasury Secretary not because it was important, but because he knew that Jefferson, Madison, Monroe and other influential democratic-republicans would resist such a federalist venture. He built such a solid foundation that when Adams got the boot, Jefferson was not only unable to do away with the bank, he was also unable to find a single illegal fault committed by Hamilton during his tenure as Treasury Secretary. Rockefeller disguised his “fail early” schemes, buying storage tanks, pipelines, refineries and byproduct plants through agents, frustrating opponents who were unable to connect him with profits, losses and laws.

Agile Release Planning, Prioritization and Tracking

The “iron triangle” of cost, time and scope is fundamental to traditional release planning. An Agile perspective on this triumvirate is expressed in the following equation:

\[ \text{Time} \times \text{Capacity} = \text{Scope} \]

where

\[ \text{Time} = \text{Number of iterations} \times \text{iteration length} \]

\[ \text{Capacity} = \text{Average velocity per iteration} \]

\[ \text{Scope} = \text{Total number of story points that can be completed in the release} \]

Using the above equation, a team with an iteration length of two weeks and an average velocity of 30 could complete 300 story points in approximately 10 iterations or 20 weeks. While seemingly straightforward, this equation must be understood within the context of the Agile approach to project scoping. [15]

It is important to note that even after stories have been broken down and estimated, they are not generally specified to the degree that would be found in a traditional requirements document. This does not necessarily imply increased risk, because successful Agile teams rely on ongoing dialogue with users, user proxies, and subject matter experts throughout the course of the release in order to gain insights needed to satisfy the users—usually better insights than could be gained from typical requirements-specification documents. If the user interaction that makes this dialogue possible is missing, the prioritization of stories across iterations is another important aspect of release planning. The following four factors are critical considerations during prioritization: [16]

- Value of the story.
- Cost of developing the story.
- Knowledge generation regarding requirements, the domain, user needs and the underlying product technology, architecture and design.
- Risk (including technology, business, schedule, cost and functionality risk).

While a certain amount of prioritization will take place during initial release planning, on Agile development projects, prioritization is ongoing and stories are often reprioritized at the end of each iteration. Successful adoption and execution of this dynamic approach to prioritization once again requires a close relationship and ongoing dialogue with program stakeholders.

One of the most commonly used charts for tracking progress on Agile releases is the release burn down chart. On the release burn down chart, the x-axis expresses iterations, while the y-axis expresses story points remaining to be completed. The chart provides an early indicator of potential future issues, but only discussions with the development team will reveal the reason for discrepancies and what actions, if any, need to be taken.

As with any other progress tracking method, using user stories to generate velocity measures can lead to some anomalous results. For example, if the user story sizes vary widely between iterations, velocity could appear lower than is warranted. This sizing difference could also result in one story taking an inordinately long time to complete, possibly even resulting in a velocity of zero. A development team should develop its own norms in terms of the relationship between the number of stories, the number of team members, and iteration duration. In other words, teams need to be agile.

For Hamilton, apart from saving the Union, his highest priority was banking. The Bank of the United States enabled the government to make good on four powers cited explicitly in the Constitution: the right to collect taxes, the right to borrow money, the right to regulate trade among states, and the right to support fleets and armies. I’m guessing very low on his backlog was meeting Vice President Aaron Burr for a duel on a lonely sandbar across the Hudson from New York. And just as Agilists argue over the explicit meaning of “processes and tools” and “interactions,” Hamilton had to deal with the Constitutional meaning of such minutiae as “necessary and proper.” In a memo to Washington in 1791 regarding the banking issue, he indicated that “necessary” didn’t mean “indispensable” so much as “appropriate.” Rockefeller’s backlog most assuredly was headed by “acquiring wealth,” since he equated wealth with his own Baptist concept of charity (that part he didn’t keep for himself).

So Do You Still Want to be an Agile Project Manager?

“The two most important days in your life are the day you are born and the day you find out why.”

—Mark Twain

At this point I feel it necessary to resurrect Hamilton and John D., having been somewhat critical. I’ll pass on Hamilton, since he’s riding a popularity crest right now with a Broadway play, avoiding being erased from the $10 bill — and hey, who doesn’t love a treasury secretary who can rap?

John D., however, is known as a robber baron and thief who enriched himself at the sake of the poor. In reality, he sold kerosene at a reasonable price to the poor, who previously sat in the dark until the sun came up. And most of what they called “crimes” in those days would be called “loopholes” today. An abolitionist who supported the Underground Railroad, he endowed the University of Chicago as the first Midwest Baptist college for their
first 10 years. At the request of nine African-American women conducting school in an Atlanta basement, Rockefeller built and continued to support in perpetuity Spelman College. He named it after his wife’s family — he was never known for putting his name on things (unlike Carnegie). As a result of losing the 1911 antitrust suit, Rockefeller found himself owning about a quarter of 34 companies instead of one. His net worth increased in that year from about $300 million to $1 billion. Exhibiting the Agile benefits of distributed cognition, those 34 companies skyrocketed in value (e.g., Standard Oil of New Jersey, New York, Indiana and California became Exxon, Mobil, Amoco and Chevron, respectively). By 1922 Rockefeller had given away over $475 million to charity — more than double what Carnegie had donated — proving that you can be Agile, wealthy and a philanthropist.

People have wanted to control things and other people since the dawn of time. If that is your goal, steer clear of Agile projects, where your role will be reduced to that of a servant leader by what is essentially an autonomous collective of self-directing and self-organizing individuals. There is currently no comprehensive project management model to understand and analyze the entire spectrum of Agile innovation, interactions and processes. After accepting the reality that one size does not fit all, practicing PMs may need to rely on a combination of models and processes to understand the extent of innovation in Agile projects and find the optimal way of managing them. If, however, you can check your ego and accept your role as a team member with honesty and integrity, you may be as successful as Alexander and John D.

REFERENCES


ABOUT THE AUTHOR

Peter D. Morris, PMP, PMI-ACP is a Process Engineering Consultant for INDUS Technology Inc. under contract to SPAWAR Systems Center (SSC) Pacific. He has worked in both the commercial and DoD/Federal sectors, including several Fortune 500 companies. He has authored numerous technical reports, including publications for the NSA, DNA, US Army NTC, USAF Space Command and the SINCGARS & JTIDS communications programs. Having managed or consulted on over 50 Agile projects in both the government and private sector, Mr. Morris has also been instrumental in establishing successful PMOs at numerous organizations, targeting strategic Business Process Improvement associated with Project and Process Management, Engineering and Project Support enhancements.
Software Process Improvement Standards and Guides for Very Small Organizations

An Overview of Eight Implementations

Claude Y. Laporte, École de technologie supérieure
Rory V. O’Connor, Dublin City University

Abstract. Very small entities (VSEs) — organizations with up to 25 people — are very important to the worldwide economy. The products they develop are often integrated into products made by larger enterprises. However, it has been established that such entities often do not utilize existing best practice standards and frameworks such as ISO/IEC/IEEE 12207 software life cycle processes standard. In addition, small organizations do not usually have the expertise to search for and adapt process improvement best practices from many frameworks to their needs. Finally, these organizations are usually also looking for low-cost evaluation or certification schemes that would provide them with visibility. To address their needs, ISO/IEC 29110 software and systems engineering standards and guides have been developed using elements of published standards. A four-stage road map has been developed to support process improvement activities of VSEs. In this paper, we present eight implementations of ISO/IEC 29110 as an exemplar of the potential benefits from the use of this standard.

Introduction

Software development is a highly complex undertaking. [1] For many small and very small software organizations, implementing controls and structures to properly manage their software development activity is a major challenge. [2] All software companies are not the same and vary according to factors including size, market sector, time in business, management style, product range and geographical location. [3] The fact that all organizations are not the same raises important questions for those who develop software process and process improvement models. To be widely adopted by the software industry, any process or process improvement model should be capable of handling the differences in the operational contexts of the companies making up that industry. But process improvement models, though highly publicized and marketed, are far from being extensively deployed. Their influence in the software industry, particularly for small and very small software companies, therefore remains at more a theoretical than practical level.

Industry, both military and civilian, recognizes the value of VSEs — i.e., enterprises, organizations (e.g., government agencies or not-for-profit organizations), departments or projects with up to 25 people — in contributing valuable products and services. A large majority of enterprises worldwide are VSEs. In addition, a large number of small projects conducted in large organizations are developed with ad hoc processes. More than ever, integrators of military systems depend on their numerous suppliers to deliver subsystems meeting evolving requirements correctly, predictably, rapidly, and cost-effectively. A supply chain of large systems often has a pyramidal structure. If an undetected defect is left in a low-level component, it may remain undetected once this component is integrated in a higher-level component.

International Standards and Guides for VSEs

The recently published set of ISO/IEC 29110 international standards (IS) and technical reports (TR) is aimed at addressing these issues as well as the specific needs of VSEs. The engineering standards and guides developed by an ISO working group, Working Group 24 (WG24), [4] are targeted at VSEs that do not have experience or expertise in selecting, for a specific project, the appropriate processes from life cycle standards such as ISO/IEC/IEEE 12207 [5] or ISO/IEC/IEEE 15288 [6] and tailoring them to the project’s needs.

A core concept at the heart of ISO 29110 is that of “profile groups” [7] that are a set of profiles. The “generic profile group” has been defined as applicable to VSEs that do not develop critical systems or critical software. The Generic Profile Group is a four-stage road map, called profiles, providing a progressive approach to satisfying a vast majority of VSEs. VSEs targeted by the “Entry profile” are VSEs working on small projects (projects that take no more than six person-months’ effort) and startups. The “Basic profile” targets VSEs developing a single application with a single work team. The “Intermediate profile” is targeted at VSEs developing more than one project in parallel with more than one work team. The “Advanced profile” is targeted to VSEs that want to sustain and grow as an independent competitive system and/or software development business. ISO 29110 is intended to be used with any life cycle, such as waterfall, iterative, incremental, evolutionary or Agile.

<table>
<thead>
<tr>
<th>Profile</th>
<th>Entry</th>
<th>Basic</th>
<th>Intermediate</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of processes</td>
<td>2</td>
<td>2</td>
<td>3 (+1 conditional)</td>
<td>3 (+3 conditional)</td>
</tr>
<tr>
<td>Number of Tasks</td>
<td>40</td>
<td>67</td>
<td>107 (+ 8 conditional)</td>
<td>120 (+ 24 conditional)</td>
</tr>
<tr>
<td>Number of Work Products</td>
<td>14</td>
<td>22</td>
<td>39 (+ 3 conditional)</td>
<td>41 (+ 5 conditional)</td>
</tr>
<tr>
<td>Number of Roles</td>
<td>3</td>
<td>7</td>
<td>8 (+ 1 conditional)</td>
<td>8 (+ 1 conditional)</td>
</tr>
</tbody>
</table>

Table 1: Processes, tasks, work products and roles of each software profile
Table 1 lists the number of processes, tasks, work products and roles of each profile of the ISO 29110 four-stage software engineering road map. A conditional process is a process that may be mandatory under some specified conditions, may be optional under other specified conditions, and may be out of scope or not applicable under other specified conditions. These are to be observed if the specified conditions apply. The “advanced” profile is not yet finalized. Therefore, the information in Table 1 is subject to change.

The software engineering “Basic” profile [8] will be used to illustrate one of the four-stage ISO 29110 road maps.

The Basic Profile for VSEs Developing Software
The software engineering Basic profile [8] defines a VSE-appropriate software implementation (SI) process and a VSE-appropriate project management (PM) process. The main reason for including project management is that the core business of VSEs is software development, and their financial success depends on successful project completion within schedule and on budget (as well as on making a profit). The high-level view and the relationships between the SI processes and the PM processes are illustrated in Figure 1.

Due to space limitation, ISO/IEC 29110 is not described in detail here. ISO/IEC 29110 has been described in more detail in a 2013 “Crosstalk” article, [9] and we suggest readers consult this. The mappings between ISO/IEC 29110 and the best practice standards (e.g., ISO 15288) are listed in an ISO 29110 information road map. A conditional process is a process that may be mandatory under some specified conditions, may be optional under other specified conditions, and may be out of scope or not applicable under other specified conditions. These are to be observed if the specified conditions apply. The “advanced” profile is not yet finalized. Therefore, the information in Table 1 is subject to change.

The software engineering “Basic” profile [8] will be used to illustrate one of the four-stage ISO 29110 road maps.

Implementations of ISO 29110
In the following subsections we will present eight implementations of ISO 29110 to illustrate the usage of this standard in a spectrum of organizational settings. Since most VSEs are using ad hoc processes, they either do not collect process measures or their process measures are not reliable. One interesting measure for the management of VSEs is the percentage of the total project effort spent in rework (e.g., correction of defects). Charette [12] reported that software specialists spend about 40 to 50 percent of their time on avoidable rework. Krasner [13] published data showing that CMM level two organizations spent 25 to 50 percent of their time on rework, while CMM level three organizations spend 15 to 25 percent. In the following implementation, we describe in detail the measures collected by one VSE, and we briefly present rework measures of two other VSEs.

Implementation 1: An IT Startup
An implementation project has been conducted in an IT startup VSE by a team of two developers. [14] Their web application allows users to collaborate, share and plan their trips simply. The use of the Basic profile of ISO 29110 has guided the startup to develop an application of high quality while using proven practices of ISO 29110. The total effort of this project was nearly 1,000 hours.

The IT startup has recorded the effort, in person-hours, spent on project tasks. Table 2 shows, for each major task, the effort needed to execute the task, the effort required to detect errors afterward, and the effort required to correct the errors (i.e., the rework).

Only 12.6 percent of total effort has been spent on rework (i.e., 125 hours/990.5 hours). This indicates that the use of appropriate standards can guide all the phases of the development of a product such that the wasted effort (i.e., rework) is about the same in a startup as in a more mature organization.

Implementation 2: A Canadian/Tunisian IT Startup
Metam is a company founded in 2013. The company has one site in Canada and one site in Tunisia. Its business domains are software development services, web solutions, mobile applications, and consulting services to implement ERP solutions. The Basic profile of ISO 29110 was used as the framework for the company’s software processes. [15] It was also used as a foundation to implement CMMI-DEV level two practices because it was requested by some military contracts. In Fall 2016, the 20-employee VSE was successfully audited by a third-party auditor against the ISO 29110 Basic profile.
**Implementation 3: A Peruvian IT Startup**

An ISO 29110 process improvement project was conducted in a Peruvian IT startup of four people. [16] After completing the implementation of the two processes of the Basic profile using an Agile approach, these processes were executed in a project with the second-largest insurance companies in Peru. Managing the project and developing the software took about 900 hours. The IT startup expended only 18 percent of the total project effort on rework. This startup became the first Peruvian VSE to obtain an ISO 29110 certification of conformity. The ISO 29110 certification greatly facilitated access to new clients and larger projects. In 2016, this VSE had 23 employees.

**Implementation 4: A Large Canadian Financial Institution**

The IT division of a large financial institution has over 3,000 employees who develop new applications and maintain more than 1,250 applications. The Cash Management IT department, which has six developers, is responsible for the development and maintenance of software tools used by traders. [17] Each year, the department is faced with more requests to add, correct, or modify features related to supported applications. Before the implementation of the ISO 29110 Agile process, customers had the following complaints:

- It is very difficult to know the status of specific requests.
- Very often, there is an incident when a change is put in production.
- There are a large number of faults detected by the quality assurance department.
- The development process is painful and the documentation produced is not very useful.

In response to these problems, processes were evaluated by comparing the activities of the actual maintenance process to those of the Basic profile. Some shortcomings were found in the actual project management process and in the software implementation process. Figure 2 illustrates the coverage of the software implementation tasks to the Basic profile before the process improvement project.

The new project management process has been adapted to the context of the division by injecting a few tasks from the Scrum methodology. The new Agile process, using the Basic profile of the ISO 29110, has been tested on three pilot projects. Recently, a five-person team was added to the TSD department to carry out all non-urgent maintenance projects using the ISO 29110 Agile process.

**Implementation 5: An Enterprise in the Automotive Field**

TM4 is a Canadian company of 140 people (including 14 software engineers) that designs and sells electric powertrain systems in the automotive field. Their products are embedded software that controls the operation of engines in real time and software that controls the interactions between the components of a vehicle.

<table>
<thead>
<tr>
<th>Title of task</th>
<th>Prevention (hours)</th>
<th>Execution (hours)</th>
<th>Review (hours)</th>
<th>Rework (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment installation</td>
<td>89</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project plan development</td>
<td>35</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Project plan execution and project assessment &amp; control</td>
<td>47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specification &amp; prototype development</td>
<td>199.5</td>
<td>7</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Architecture development</td>
<td>42.5</td>
<td>1.5</td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>Test plan development</td>
<td>12.5</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Code development and testing</td>
<td>361</td>
<td>47</td>
<td>96.5</td>
<td></td>
</tr>
<tr>
<td>Develop user guide &amp; maintenance document</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Web site deployment</td>
<td>8.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project closure</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total hours</strong></td>
<td><strong>89</strong></td>
<td><strong>716</strong></td>
<td><strong>60.5</strong></td>
<td><strong>125</strong></td>
</tr>
</tbody>
</table>

*Table 2: Effort to execute a task, then detect and correct errors by the two-member team (from [14])*

**Implementation 5: An Enterprise in the Automotive Field**

The new project management process has been adapted to the context of the division by injecting a few tasks from the Scrum methodology. The new Agile process, using the Basic profile of the ISO 29110, has been tested on three pilot projects. Recently, a five-person team was added to the TSD department to carry out all non-urgent maintenance projects using the ISO 29110 Agile process.

**Implementation 6: A Large Utility Provider**

The IT division of a large Canadian utility provider has 1,950 employees that support more than 2,100 software applications. The organization had already implemented 12 level two and three process areas of the CMMI-DEV. Traditional life cycles were used for the development of this division.

A small department within the IT division, the Mobility and Georeferenced Solutions Department, is composed of six developers, three analysts, one architect, and one manager. Typical projects of the department are requests from internal customers to improve a few applications. The department was required to develop applications more quickly, and with very different technologies. Increasingly, the department had to develop proof of concepts. The problem was that the deliverables requested by the current methodology for typical projects of the IT division were so numerous that the level of documentation required was not suitable for small projects and small teams.
A project was launched within the small department to tailor ISO 29110 to their needs and to adapt it to a Scrum approach. A pilot project was conducted that involved the creation of a web application for property management. This application greatly facilitated geographic data consultation. The total effort of this project was about 1,500 hours. About 8.5 percent of the effort was invested in prevention tasks, while only 9.6 percent was spent in rework tasks. The ISO 29110 process improvement project allowed the small department to shine within the IT division as it became a model for future small IT projects.

Implementation 7: Implementation in a Medical R&D VSE

A project has been conducted to develop and implement a quality management system for a medical R&D company of 15 employees [19]. The VSE manufactures a family of neuronavigation products that are used in over 400 laboratories around the world in the fields of cognitive neuroscience, rehabilitation research and veterinary sciences.

This project improved the business processes and implemented a quality management system in accordance with the ISO 13485 medical standard.

This project used the ISO 29110 systems engineering Basic profile [20] to facilitate the implementation of ISO 13485. ISO 29110 has guided the VSE in the development of tools, guides and templates. During this project, totaling more than 1,000 hours of effort, the implementation of the quality system was planned; processes, guides and templates were defined in collaboration with key resources of the company. A pilot project was conducted to validate the adequacy of the established process.

The use of ISO 29110 systems engineering Basic profile facilitated the implementation and the adaptation of a standard such as ISO 13485 for the VSE.

Implementation 8: A Small Canadian Company in Public Transportation

This project was created to define and implement project management and systems engineering processes at CSinTrans Inc. (CSiT), a Canadian company created in 2011. [21] The company specializes in the integration of interactive systems, communication, and security in the field of public transportation. The recently published ISO 29110 management and engineering guide for systems engineering [20] has been used as the main reference for the development of their processes.

ISO 29110 was a good starting point to align its processes with the CMMI-DEV since conformity to the CMMI-DEV is a requirement from some customers in the transit industry. To better respond to different types of projects, CSiT developed three process groups (i.e., light, standard, and full), each being adapted to meet certain attributes of projects, such as size and type (e.g., prototype, typical project).

ISO 29110 standard has helped raise the maturity of the young organization as the organization has implemented proven practices and developed uniform work products. ISO 29110 has also helped in developing lightweight processes, allowing the young company to remain flexible and maintain its ability to react quickly to its customers. In mid-2016, the systems engineering Basic Profile of the ISO 29110 was successfully audited by a third-party audit team composed of two independent auditors. One member of the audit team was a systems engineering domain expert.

Conclusion and future work

The eight implementations presented in this paper have demonstrated that using ISO/IEC 29110 made it possible to properly plan and execute projects, develop products, and conduct projects using proven system or software engineering practices. This disproves the perception that a process standard interferes with the creativity of software developers. Rework data collected from three of the VSEs (two of which were startups) was presented. This provided some evidence that the implementation of proven practices documented in ISO/IEC 29110 allowed them to execute their projects with a percentage of rework similar to CMMI® level 3 organizations. The relationship between the success of a software company and the software process it utilized had been investigated, showing the need for all organizations — not just VSEs — to pay attention to software process practices such as ISO standards.

The eight implementations described are a small subset of implementations around the world. For instance, in Thailand over 320 private organizations and 15 public organizations have been ISO 29110 certified against the Basic profile, in Mexico 33 organizations have been certified against the Basic profile. [22] Recently, a government agency launched a first program to certify 110 VSEs against the Basic profile.

Since a few ISO 29110 documents, such as the management and engineering (M&E) guides are freely available and have been translated in Czech, French, Portuguese, Spanish and adopted as national standards, the number of implementations should increase. Also, since the ISO 29110 M&E guides are freely available and easily understandable, more than 15 countries are teaching ISO 29110. For instance, in Thailand, 10 universities are teaching ISO 29110.

The software Intermediate profile has been recently published. [23] The Advanced profile should be published in 2018. The development of the systems engineering Intermediate and Advanced M&E guides has started in mid 2017. WG24 has also been mandated to develop an ISO 29110 service delivery guide as well as a standard to help VSEs in providing services delivered to internal or external customers.

Finally, since many VSEs developing systems are also involved in the development of critical software or systems, WG24 will conduct an analysis to determine if a set of ISO 29110 systems/software M&E guides and standards, for VSEs developing critical products, should be developed.

Additional Information

The following website provides more information as well as articles and deployment packages for software and systems engineering: http://profs.logi.etsmtl.ca/claporte/English/VSE/index.html
REFERENCES


ABOUT THE AUTHORS

Dr. Claude Y Laporte, P. Eng., has worked in defense and transportation organizations for more than 20 years. Since 2000, he has been a professor at the École de technologie supérieure, an 11,000-student engineering school in Montréal, where he teaches software engineering. He is the project editor of ISO/IEC 29110 set of systems and software engineering Standards and Technical Reports. He is the co-author of a software quality assurance textbook that is anticipated to be published by the IEEE Press-Wiley in 2017. Website address: http://profs.logi.etsmtl.ca/claporte/English/index.html

École de technologie supérieure
Department of Software and IT Engineering
1100, Notre-Dame Street West,
Montréal, Québec, Canada, H3C 1K3
Email: Claude.Laporte@etsmtl.ca

Professor Rory V. O’Connor is a professor of software engineering at Dublin City University (Ireland) and a senior researcher with Lero, the Irish Software Research Centre. He is Ireland’s Head of Delegation to the International Organization for Standardization (ISO) for Software & Systems Engineering (ISO/IEC JCT1/SC7). His research interests are centered on the processes and standards whereby software intensive systems are designed, implemented and managed. Website address: http://www.roryoconnor.com

Lero, Irish Software Engineering Research Centre,
Dublin City University, Dublin, Ireland
Email: Rory.OConnor@dcu.ie
Leveraging the Power of Historical Data Through the Use of Trend Lines

Taylor Putnam-Majarian and John A. Staiger, Jr.

Abstract. Developing software within the DoD presents a unique set of challenges, including but not limited to budget cuts, Congressionally mandated changes, changing software requirements, and so on. It should come as no surprise, therefore, that cost estimators have faced significant challenges when estimating systems in the Defense arena. A recent initiative put forth by the DoD was to improve its estimation process by leveraging historical data collected from forensic analyses of recently completed software development efforts. This article discusses (1) some of the challenges faced throughout this initiative, (2) the data collection process, and (3) how one can leverage data to improve cost estimates.

Developing software within the Department of Defense (DoD) presents a unique set of challenges. The environment is such that many of the programs are unusually large, complex, and schedule-driven, and typically require several years to develop, field, and deploy. Often, they are maintained for over a decade, during which time several advancements in technology, unknown at the inception of the program, must be accommodated into the existing system. It should come as no surprise, therefore, that cost estimators have faced significant challenges when estimating systems in the defense arena.

The Quantitative Software Management (QSM) team has supported the DoD at a variety of levels, from program management offices to the Secretary of Defense. One such organization’s recent initiative was to improve its estimation process by leveraging historical data collected from forensic analyses of recently completed software development efforts. This article discusses (1) some of the challenges faced throughout this initiative, (2) the data collection process, and (3) how one can leverage data to improve cost estimates.

Challenges

The Spanish philosopher George Santayana is credited with the observation that “those who do not learn from the mistakes of history are doomed to repeat them.” This astute piece of wisdom has been paraphrased and quoted many times, but it is especially relevant in the DoD environment due to the unique challenges posed by the size and complexity of DoD systems. In order to learn from past mistakes, it is important to review the specific actions, assumptions, and information that were used and available. Thinking back to a completed project that was considered successful, what were some characteristics that allowed it to do well? Conversely, thinking back to a project that was considered less than optimal, what should have been done differently? This type of information is often readily available informally, but it isn’t often captured in a quantifiable form. Hence historical data should be collected in order to objectively assess processes and quantify actual performance.

Collecting historical data, however, can sometimes be a sensitive subject. It is not uncommon for government organizations to face some resistance during the initial phases of data collection. One of the main challenges faced is a fear of measurement. Stated simply, people do not want to be viewed, either by their peers or their superiors, as less than successful. Even when projects have performed well, some still believe that any data disclosed could potentially be used against them. If we are able to overcome this first challenge, we’re often faced with a second, which is that no data is available.

Finally, historical data can be — and often is within the DoD environment — competition-sensitive. These challenges are typically navigated by using data from third-party industry benchmarks of completed programs rather than the organization’s actual historical performance data. If better data is unavailable or confidential for contractual reasons, a subset of data from comparable projects is used in lieu of client data.

Before discussing the process of how historical data is collected, it is important to first discuss why this information should be collected in the first place. Historical data serves as the foundation for quantitative, data-driven estimating, tracking, and process improvement. First and foremost, collecting historical data promotes good record keeping. While data collection is often a standard practice among DoD systems, it is most often decentralized or not properly validated, making it difficult to find the exact piece of information needed without searching through multiple documents or having a very low confidence level attached to it. Consolidating historical data into one location with appropriate levels of protection for each program makes it easier and more efficient to reference this information later. Second, historically, data can be used to baseline and benchmark other historical projects in order to determine which projects were completed successfully and which could have been improved. Finally, and very important for DoD “budgeteers,” historical data can be leveraged for producing defensible, quantitative estimates for future projects. The following sections will discuss each of these points in greater detail.

Collecting Historical Data

How does one go about collecting historical data? Beginning this process can be intimidating, especially if historical data has not previously been collected. In order to simplify the process, one should instead focus initially on collecting just a few key pieces of information, adding more detailed metrics to the repository later.

For each phase of development, the start and end dates are collected and the schedule duration is calculated. This is most often represented in months, but weeks are sometimes used if the project is especially short or if Agile development is used. Next, data on the effort expended is gathered. This is most often represented in person-hours or person-months (or man-month/staff-months). The peak number of staff working on the program is also recorded. The last necessary piece of information is...
project size. This is typically captured in the form of source lines of code, function points, business or technical requirements, Agile stories, or RICEFW objects. Together, these pieces of information can be used to examine the project holistically.

Where can the required data be found? When beginning the data collection process, it is important to identify potential data sources or artifacts. This part of the data collection process can be similar to an archaeological dig. There will be a lot of data and information, but not all of it will be relevant or useful. However, it is necessary to sort through all of it in order to determine which pieces are relevant and necessary. In some cases, data must be acquired indirectly or derived using the artifacts and information that is available, particularly if the primary data is missing, unavailable, of questionable confidence, or if confirmation of key metrics is needed. Typically, requirements documents contain some early sizing information, and code counts can be run at the end of the development effort to capture a more refined sizing measure. Software Resource Data Reports (SRDRs), Cost Analysis Requirements Descriptions (CARDs), Automated Cost Estimating Integrated Tools (ACEIT) files, and program briefings are all useful for collecting schedule and effort data. Moreover, if vendors are required to submit their data, Microsoft Project© files, Clarity©, and other PPM© exports can be additional valuable data sources to verify staffing, schedule, and effort.

It is beneficial to have multiple source options for obtaining information. Many of these documents will undergo several iterations and may be updated at each milestone of the acquisition life cycle. Unless the data appears questionable or invalid, it is usually best to use the data from the most recent version of the source document. Having multiple source options allows for validation, derivation of key metrics, and contextual information, all of which serve to maintain the integrity and increase the value of the historical performance database.

**Putting Historical Data to Use**

As an example, QSM’s support team went through the process of collecting and validating a set of data from recently completed Defense Major Automated Information System (MAIS) projects. Within the DoD, MAIS programs are those systems consisting of computer hardware, computer software, data, and/or telecommunications that perform functions such as collecting, processing, storing, transmitting, and displaying information. They are also above a specific cost threshold or designated as a special interest program. With this data set, the team was able to create a set of MAIS trend lines as shown below (see Figure 1).

Although these 16 data points came from three distinct vendors, notice that the r2 value equals 0.76, indicating that the relationship between size and productivity for these releases was highly correlated. This type of correlation is typical when examining data points from one organization. The strength of this correlation, particularly with three different vendors, indicates that this trend line would be appropriate for analytical use.

Note that Figure 1 demonstrates the correlation between size and productivity. Additional trend lines and correlations can be developed for numerous other metrics, such as effort, duration, defects and staffing levels. The development of trend lines is limited only by the data that is collected.

**Using Historical Trend Lines**

Once a set of custom trend lines has been developed, there are two primary ways in which they can be used: 1) analyzing and benchmarking completed projects, and 2) estimating future releases. Each of these will be discussed further in the sections below.

**Assessing Past Performance**

The use of historical trend lines can be instrumental in accurately assessing the performance of completed projects. Figure 2 shows the staffing trends for Defense MAIS programs, plus or minus three standard deviations from the mean. The data points plotted on this chart represent notional completed projects submitted by a notional contractor.
When compared with other Defense MAIS systems, this contractor is using significantly more people to complete its releases. Upon initial examination, this chart raises some questions. Why did this contractor need so many people when other vendors were able to deliver the same functionality with a fraction of the staff? Might there be a reason for this behavior? Perhaps the contractor was behind schedule and hoped that adding more people would help the team complete the program on time. Was this a time and materials or a fixed price contract? Perhaps this contractor used large teams as a means of billing more work if the contract type was favorable to this strategy. Perhaps the contractor was behind schedule and hoped that adding more people would help complete the program on time. Was this a time and materials or a fixed price contract? Perhaps this contractor used large teams as a means of billing more work if the contract type was favorable to this strategy.

These types of analyses can shed light on the various factors that might have impacted project performance and can give clues as to what adjustments can be made to improve processes in future projects. This same technique can also be applied to the vendor selection process. If all potential vendors were required to submit data from comparable completed projects, the acquisition or source selection office would be able to plot these data points against industry trend lines to validate the submitted bids. This would serve to determine the likelihood that the vendors would be able to complete the work according to their proposals and historical performance. It would also help determine whether the bids were feasible against the government's independent (or ‘should cost’) estimate and within its expected schedule and budget. Quite obviously, this serves a valuable purpose for acquisition, budget, and oversight offices alike.

**Estimation**

Aside from benchmarking, the other main use for historical trend lines is in estimating future releases. Before getting into specifics, it is important to first understand the best practices for project estimation. At the beginning of a project, when the least information is known, a top-down method is the quickest and easiest way to create an estimate. Several parametric vendors utilize this top-down approach to software estimation as well, though the assumed inputs and algorithms may differ somewhat among the various tools.

For example, QSM uses a proven production equation that is constantly adjusted and refreshed through the analysis of a very large set of actual historical data. The software production equation is composed of four terms, namely: size, schedule, effort, and productivity. Productivity is measured in index points,

**WE ARE HIRING**

**ELECTRICAL ENGINEERS AND COMPUTER SCIENTISTS**

As the largest engineering organization on Tinker Air Force Base, the 76th Software Maintenance Group provides software, hardware, and engineering support solutions on a variety of Air Force platforms and weapon systems. Join our growing team of engineers and scientists!

**BENEFITS INCLUDE:**

- Job security
- Potential for career growth
- Paid leave including federal holidays
- Competitive health care plans
- Matching retirement fund (401K)
- Life insurance plans
- Tuition assistance
- Paid time for fitness activities

Tinker AFB is only 15 minutes away from downtown OKC, home of the OKC Thunder, and a wide array of dining, shopping, historical, and cultural attractions.

Send resumes to:
76SMXG.Tinker.Careers@us.af.mil
US citizenship required
etary QSM unit which ranges from 0.1 to 40. It allows for meaningful comparisons to be made between projects and accounts for software development factors, including variables such as management influence, development methods, tools, experience levels, and application type complexity. This equation is described below:

\[
\text{Size} = \text{Productivity} \times \text{Time}^{4/3} \times \text{Effort}^{1/3}
\]

The exponents attached to the time and effort variables differ significantly. The schedule variable, or time, is raised to the 4/3rd power while the effort variable is raised to the 1/3rd power. This indicates that there is a diminishing return on schedule compression as staff is added to a software project. In extreme conditions, this indicates that consuming large amounts of effort does little to compress the project schedule. This equation can be rearranged algebraically to solve for any of the above variables.

Using a top-down parametric estimation tool makes it possible to quickly create an estimate very early in the life cycle, when little information is available. If the size and productivity assumptions are known (e.g., from similar past performances), then it is possible to calculate the duration and the amount of necessary effort. Early sizing methods can be based upon analogous program efforts, requirements, or function points and can be refined over time as more information becomes available. The size-adjusted productivity values can be taken from appropriate trend lines based on historical performance. This very technique has been used to successfully estimate and model future releases of Defense MAIS programs.

**Refining the Cycle**

One of the major advantages of having access to historical data is that it can be used at any point during the software development life cycle. Figure 3 below describes this cycle:

When a project is in its early phases, historical data can be leveraged to create an estimate based on the developer's demonstrated performance on other completed projects. Once accepted, this estimate can serve as the baseline plan for the program. Once the project is underway, actual data can be collected at regular periodic intervals (e.g., monthly) and tracked against the estimate to assess and characterize actual performance. If the actual performance deviates from the estimated plan, it is possible to reforecast the schedule duration and necessary effort (i.e., staffing) values. Upon completion, the final data can be collected and added to the historical data repository. An updated set of historical trend lines can be produced and can then be used for estimating future releases. Over time, you will begin to see that there is less discrepancy between the estimates and the actual outcomes (excluding the impact of external factors, such as changed or additional requirements, budget cuts, Congressionally mandated changes, or other factors unique to the Defense environment), thus refining the cycle to improve processes across the entire software development life cycle. However, more accurate estimation of factors that can be controlled certainly helps mitigate the overall impact of those factors that cannot. Estimation can also be done at an early stage, when the cost of adjustments is far less than later in the development process.

**Summary**

The British mathematician Charles Babbage stated that “errors using inadequate data are much less than those using no data at all.” This quote really speaks to the advantage that can be found in collecting an organization’s data or in the use of data available in the industry. As more programs complete, an organization can continue to add to its historical database, refine its estimates, and improve processes throughout the entire software development life cycle. With the assistance of an IT cost analyst, these process improvements — particularly as demonstrated within the DoD — can lead to significant savings in effort and schedule.

**ABOUT THE AUTHORS**

**Taylor Putnam-Majarian** is a Consulting Analyst at QSM and has over seven years of specialized data analysis, testing, and research experience. In addition to providing consulting support in software estimation and benchmarking engagements to clients from both the commercial and government sectors, Taylor has authored numerous publications about Agile development, software estimation, and process improvement, and is a regular blog contributor for QSM. Taylor holds a bachelor’s degree from Dickinson College.

**John A Staiger Jr.** is a Principal Consultant and Trainer for QSM and has worked with SLIM tools since 1985. He has extensive experience in data analysis and parametric methods. He earned a BA in Economics from the University of Michigan, an MBA and Post-MBA certificate in Management Science from Seattle University and an M.S. in Project Management from The George Washington University. He is also a Distinguished Graduate of the Navy War College.

**Figure 3. Data Refinement Cycle.**

**Taylor Putnam-Majarian** is a Consulting Analyst at QSM and has over seven years of specialized data analysis, testing, and research experience. In addition to providing consulting support in software estimation and benchmarking engagements to clients from both the commercial and government sectors, Taylor has authored numerous publications about Agile development, software estimation, and process improvement, and is a regular blog contributor for QSM. Taylor holds a bachelor’s degree from Dickinson College.

2000 Corporate Ridge, Suite 700
McLean, VA 22102
703-749-3551
taylor.putnam-majarian@qsm.com

John A Staiger Jr. is a Principal Consultant and Trainer for QSM and has worked with SLIM tools since 1985. He has extensive experience in data analysis and parametric methods. He earned a BA in Economics from the University of Michigan, an MBA and Post-MBA certificate in Management Science from Seattle University and an M.S. in Project Management from The George Washington University. He is also a Distinguished Graduate of the Navy War College.

2000 Corporate Ridge, Suite 700
McLean, VA 22102
703-790-0055
john.staiger@qsm.com
Debunking Agile Myths

by Dick Carlson

It is often interesting and sometimes amusing how often people believe “facts” that are not true. Misinformation, confusion, and a lack of practical experience create challenging barriers to the acceptance and application of Agile and Lean, to software and systems engineering techniques. Since the Agile Manifesto for Software Development was developed in 2001, large and small companies alike have experienced an increase in productivity, improvements in employee satisfaction, and higher customer satisfaction when Agile practices are used. [1]

This article discusses and analyzes 10 of the most common misconceptions about Agile practices and techniques and identifies real-world examples that debunk these misconceptions. Included are actual examples of successful applications of Agile that have resulted in the development and deployment of deliverable software and other products within a wide range of businesses and industries.

The misconceptions to be addressed in this article include:

1. Agile methods are undisciplined and not measurable.
2. Agile methods have no project management.
3. Agile methods apply only to software development.
4. Agile methods have no documentation.
5. Agile methods have no requirements.
6. Agile methods only work with small co-located teams.
7. Agile methods do not include planning.
8. Agile only works for small project teams.
9. Agile development is not predictable.
10. Agile development does not scale.

AUTHOR’S NOTE: The company’s name is protected by a very strict code of ethics regarding specific business practices and the identification of its customers. Therefore, references to the company, its customers, and the name of any program cannot be disclosed.

My apology to those readers who believe that a more important misconception has been omitted in order to include a lesser misconception. This list is based on my own personal experiences and opinions.

Applications of Agile

The application of Agile to projects has resulted in higher success rates over many years because of its specific practices and principles, including:

- Simplicity.
- Short iterations.
- Embracing change.
- A sustainable pace.
- Customer satisfaction.
- Verifiable collaboration.
- Daily stand-up meetings.
- Prioritizing requirements.
- Continuous improvement.
- Time-boxed work sessions.
- Close customer collaboration.
- Frequent planning and estimating.
- Frequent releases of working features.
- Product demonstrations and artifact reviews.
- Motivated, self-organized and cross-functional teams.

Agile Myths

The following are myths collected from my personal experience and from the experience of others. All are presented in a relative order of popularity.

1. Agile methods are undisciplined and not measurable.

This myth is among the most common and the most damaging when attempting to explain an Agile approach to an uninformed executive or customer. Though many people attempt to find discipline and predictability in traditional projects, these projects often involve extensive processes, excessive meetings, and extensive data management that do not achieve the desired discipline and predictability.

Agile projects use a different approach that involves a disciplined method with a diminutive focus. Some people are unaware that a “disciplined method” is involved. Agile projects are conducted through a series of short iterations (aka “sprints”), each of which requires teams to:

- Establish a goal or set of goals.
- Make a plan and determine what to build during each sprint.
- Estimate the effort required to build each item.
- Decompose the items into quantifiable tasks.
- Establish a team commitment to sprint goals.

This means that team members, sponsors and managers must commit to the sprint and that team members are expected to maintain focus throughout the duration of the sprint. Now this is discipline!

The evidence debunking this myth includes the completion of a multitude of projects that have implemented Agile rigorously with positive, measurable results. Further evidence in the VersionOne 10th Annual State of Agile Survey revealed that the top three benefits of Agile include:

1. An ability to manage changing priorities: 87 percent.
2. Increased team productivity: 85 percent.
3. Improved project visibility (transparency): 84 percent.

The survey also showed that the leading causes of failed Agile projects during 2015 were:

1. A company philosophy or culture at odds with core Agile values: 46 percent.
3. An absence of management support: 38 percent.

Three of the leading reasons indicated by “How Success is Measured with Agile Methods” included:

1. On-time delivery: 58 percent.
3. Customer/user satisfaction: 46 percent.

2. Agile methods have no project management.

This myth was likely formed by those unfamiliar with Scrum, which is an Agile project management approach that focuses on delivering the highest business value first. Scrum is implemented through a series of short sprints (two to four weeks in duration), with each sprint producing an increment of potential functionality. After each sprint, decisions are made to either
release the product as is or to continue enhancing the product until it is ready to be delivered or deployed. Scrum is simple and straightforward, and its activities, practices and rules are few and easy to learn. Scrum minimizes project planning because team members select their own work and self-organize.

While most projects have a project manager or someone responsible for project requirements and expenditures, the evidence debunking this myth is supported by the global implementation of Scrum. Scrum is an Agile project management framework that includes three roles (the development team, the product owner, and the Scrum master), four activities or working sessions (sprint planning, the daily Scrum or standup, the sprint review, and the sprint retrospective), and a few work products including the product backlog, sprint backlog, task board, burn-up chart, operational feature, and/or working software. The Scrum master facilitates all working sessions, arranges facilities, ensures all team members are fully functional and productive, and makes certain that everyone follows the agreed-upon Scrum process. The product owner collaborates closely with stakeholders for release planning and manages the product vision, project requirements, product road map, and product backlog. The team plans for, builds and manages chosen deliverables during each sprint.

Further, problematic processes do not achieve wide acceptance. Agile methods continue to receive wider acceptance.

3. Agile methods apply only to software development

This myth is most likely based on a combination of the principles expressed in the Agile Manifesto for Software Development used in most Agile projects. [2] Only three of these principles apply specifically to software development. The remaining nine principles are generic and apply to just about any kind of project, circumstance, or situation that must be managed. In effect, the remaining principles can be used in anything we do. Whether they are applied on the job, at home, or in our communities, a majority of Agile principles can be used in almost any scenario.

One case with evidence debunking this myth is how quickly a five-day, comprehensive Agile-training course was developed in 60 days when it was originally anticipated to take six months. The project started with six people developing a 12-module course. After the first few weeks, the training deck contained more than 550 slides, many of which were redundant. As most team members were pulled back into other projects and support diminished, it became apparent that our small team of three had to rethink its approach. The solution was a no-brainer. We applied Scrum practices that reduced the six-month effort to 60 days and reduced the training deck from more than 550 slides to 350 slides. When presented, the course was very effective, and the course developers and instructors received kudos for the quality of the training.

Another example is a large project that used the Scrum project management approach. At the beginning of the project, it had failed to create several critically needed product backlogs across several engineering domains that should have contained enough user stories to define the operational capabilities and functionality of the application under development. Thus, several domain-related sub-projects were established. Each of these sub-projects planned and executed a series of short sprints, each with a goal to develop requirements in user story format and create product backlogs so software development teams could develop software and write tests based on those requirements. The results of this effort produced hundreds of user stories that were written and prioritized in each of the product backlogs so that the software development teams were able to use the backlog to build a wide array of functionality.

Another example is a project on a large defense program that was chartered to develop detailed specifications for equipment racks that was outsourced to an offshore company to build test benches. Management knew that projects of this type typically ran over budget and failed to meet schedules, so they looked to the software engineering effort that was using Scrum to better manage the project. Project stakeholders decided to use a seasoned Scrum master and identified a product owner who possessed the requisite technical domain knowledge, understood the challenges of the project, and was available to support the project’s life cycle that was limited to 12 months. The result was that the project finished ahead of schedule by four months and saved 20 percent of the projected costs.

4. Agile methods have no documentation

This myth was formed from a misunderstanding of the Agile Manifesto principle that states, "Working software over comprehensive documentation." To some, this means "no documentation," but to others it means that documentation specified by contract will be developed and delivered as agreed upon, but will be developed incrementally — just like other work products specified in the contract.

Evidence that debunks this myth comes from hundreds of projects where contractual documentation was developed and delivered (or is in development and will be delivered). These documents follow the same life cycle pattern as all other project deliverables except that they are delivered incrementally to avoid the risk of developing product features that may exceed cost thresholds or never be used.

Scrum is used to manage hundreds of projects that are required to develop suites of documentation, including software and systems requirements specifications, software and system design specifications, software test descriptions, system validation descriptions, software and system user manuals, training manuals, and other documents. This is not unusual, and most of these projects are capable of preparing these documents incrementally while the products are being built.

5. Agile methods have no requirements

This myth may be a consequence of misunderstanding the Agile Manifesto, or it may have been originated by someone with limited software or system development experience.

The evidence against this myth is that all software and system development methods start with the definition and development of requirements. After all, what does one build if there are no requirements? For the most part, Agile projects — especially those implementing Scrum or Extreme Programming — convert known requirements into user stories, which are descriptions of desired functionality or an aspect of a feature told from the perspective of the user or the customer. Stories focus on the "what," not the...
For example, all distributed teams face unique challenges that arise from the loss of rich social and physical interactions. Another challenging aspect is the transition from a face-to-face environment to an environment where video conferencing, instant messaging, email, shared work areas (repositories), shared calendars, and workflow and collaboration tools replace the convenience of a physical working environment. In spite of these challenges, we find ways to adapt and perform.

At one time this myth was more true than false. The Agile Manifesto was established in February 2001. In the 16 years that have passed since then, our communication capabilities have increased immensely. The capabilities of WebEx, Skype, and GoToMeeting barely existed in 2001. We have learned and experienced much since 2001. While face-to-face communication remains the best method, new tools make telecommuting feasible and cost effective. A team in Burbank, California, can work with a team in Bangalore, India. It has been done effectively.

**7. Agile methods do not include planning**

This myth may come from a major misunderstanding of the Agile Manifesto, or perhaps from a rumor of an Agile project gone bad. Were backlogs and parking lots created? Were burn-down and burn-up charts used? Were daily stand-ups held? All of these artifacts and activities are part of planning and controlling.

Rumors result when those who do not practice Agile or do not understand it form negative opinions of activities without empirical experience. Poor Agile projects include those that failed to yield favorable results.

Evidence against this myth is in the application of Scrum on Agile projects. Scrum includes a significant planning effort that begins with the creation of a vision, a product road map, some release planning, and some sprint planning. Creation of the vision, product road map, and release planning are conducted during Sprint Zero (aka Iteration Zero, initial planning, or something similar) prior to any project activity and is not usually time-boxed like other Scrum activities. Sprint Zero is a term meaning a planning session that takes place before project execution. Sprint Zero is executed at a high level and may include the customer, end-users, and select project and support personnel.

Sprint planning is conducted on the first day of every sprint and is typically time-boxed at four to eight hours. In sprint planning, the product owner and the team determine what must be completed during the sprint. The team estimates all items targeted for a sprint’s completion, selects a certain number of product backlog items, then determines how to build those items during the sprint. Clearly, this is planning!

**8. Agile only works for small project teams**

This myth is unfounded as projects using Agile practices are conducted every day around the world by both small teams and, many times, by much larger teams.

The evidence against this myth is the lack of knowledge and empirical experience on the part of the myth’s originators. The Scrum framework recommends small teams of five to nine self-organized and cross-functional people who possess software designing, coding, and testing skills. Teams staffed with...
10 to 15 systems and software engineers are quite common, although it is recommended that larger teams be divided into smaller, synchronized teams.

Further evidence of this unconfirmed myth can be explained by the Disciplined Agile Delivery (DAD) life cycle model, where the focus is on the delivery portion of the system life cycle from a project’s beginning to its release. [3]

9. Agile development is not predictable

This myth is debunked by the fact that many of the teams that use Agile practices and principles every day experience a significant increase in productivity and work output due to their strong commitment and focus. Regardless of the challenge or product being developed, planned, and estimated, the end goal is always the same — a potential increment of working software at the end of each sprint.

Evidence debunking this myth includes the following:

—Agile efficiently and effectively replaces detailed, speculative plans with high-level, feature-driven plans that 
  acknowledge the inherent complexity and uncertainty of software development projects.

—Ongoing reconciliation of actual effort to original plans is replaced with incremental planning and re-planning with 
  smaller granularity throughout development activities. Reduced granularity results in increased insight.

—Agile development operates in a rapid, iterative fashion, so valuable historical data quickly emerges for supporting 
  both short- and long-term planning.

10. Agile development does not scale

This myth does not consider that, generally speaking, software development has scaling issues. Evidence against this myth is that this is not a method-specific problem. The larger a project’s scope, the greater the probability of failure; the greater the number of people involved in a project, the greater the communication complexity and, therefore, the risk. Agile development accepts these realities and recommends smaller projects, shorter delivery times, and smaller teams. Smaller teams have proven to be much more productive than larger teams.

Agile methods promote taking large projects and decomposing them into a coordinated series of smaller projects staffed by smaller, motivated, self-organized, and cross-functional teams. Work output is integrated at least every sprint in order to reduce risk and ensure functional and technical compatibility. The recommendation is to take, where possible, a large project and decompose it into smaller subprojects. Creating these smaller subprojects increases the probability of success.

For many years, a significant number of Agile projects involving hundreds of people have been conducted by multiple teams, in multiple locations, and across multiple time zones while experiencing a high degree of confidence in the ability of the Agile development process to scale appropriately. If a com-
company has a very large, complex problem to solve, there are many reasons to prefer the use of an Agile approach to expose risks quickly, demonstrate business value early, and institutionalize a highly disciplined approach to software development.

For more than three years, I assisted and coached a very large program within a very large, multi-national company. My assistance helped to transform the company’s traditional development methods to Agile. The results were a reduction in schedule by more than 60 percent and a cost reduction of just under 40 percent. The organization consisted of 5,000 personnel that included software engineers, system engineers, and a large group of highly skilled assemblers and construction technicians.

**Conclusion**

As previously stated, the aforementioned myths are collections of Agile myths that have been debunked through personal experience. Further searches on this topic will reveal different myths reported by others in the field. To recap the myths reported in this article:

“Agile methods are undisciplined and not measurable” is disproved by the multitude of projects, past and present, that depend on commitment and focus on the part of everyone involved. It is also disproved by measured evidence in terms of cost and time savings.

“Agile methods have no project management” is disproved by the fact that everyone committed to an Agile project is taught to be self-managing. It is also disproved by the many projects that are using Scrum as a project management approach.

“Agile methods apply only to software development” is disproved by the many projects around the world that are using Scrum to manage the development of products unrelated to software. I have coached and facilitated numerous teams that produced nonsoftware products.

“Agile methods have no documentation” is disproved by the fact that documentation is produced on virtually all projects, regardless of whether it is contract-specified or project-generated. Essentially all of the teams that I have coached and facilitated over that last 15 years have produced value-added documentation.

“Agile methods have no requirements” is disproved by its ludicrous implications. Products are not developed through osmosis; products are developed from creative thinking that is defined and developed through extensive analysis and documented as specific requirements.

“Agile methods only work with small, co-located teams” is debunked by the fact that there are hundreds of Agile projects functioning with geographically distributed team members. I have coached and facilitated many teams that were not co-located.

“Agile methods do not include planning” is debunked through planning sessions that take place at the start of every sprint, during the daily stand-ups, and prior to project start during Sprint Zero, where all project and release planning activities are conducted. Projects that do not plan are doomed to fail.

“Agile only works for small project teams” is debunked by my own experience. I have coached and facilitated teams as small as two people and as large as 13 people. Very small teams are productive but cannot complete as much as larger teams can. However, I maintain through experience that teams of five to seven members are by far the most productive when developing software products.

“Agile development is not predictable” is debunked by evidence of progress and success throughout an Agile project’s life cycle through rigorous, rapid, and incremental planning, iterative and evolutionary development, and the application of simple metrics.

“Agile development does not scale” is debunked by the evidence that smaller teams are more productive than larger teams, and through Agile’s highly collaborative nature, identifies issues and risks early and drastically reduces their impacts through viable mitigation strategies.

**REFERENCES**

- Manifesto for Agile Software Development (http://agilemanifesto.org/).
- Scaled Agile (http://www.scaledagile.com).
- Scott Ambler (http://www.ambysoft.com/scottAmbler.html).
- Software Engineering Institute, Carnegie Mellon University.

**Recommended Reading:**


**NOTES**

1. The Agile Manifesto appears on the Agile Alliance website at http://agilemanifesto.org/

**ABOUT THE AUTHOR**

Dick Carlson has a Bachelor of Science degree in business management. He has held certifications as a Scrum Professional, Scrum Master, and Scrum Product Owner, and in Lean-Agile Project Management. He is an accomplished software engineering process analyst, and has shared successful experiences of Agile, Lean, and Scrum implementation at conferences, workshops, and symposia. Dick's engineering career spans more than 40 years, and he has taught courses in mathematics, electronics, CMMI, configuration management, data management, Agile, Lean, and Scrum for more than 30 years.

dcarlson@iascar.us
Which Agile Method is the Best?

By Earle Soukup

Agile methods are considered to be a light form of project management because there are fewer and less complex prescribed practices than those in traditional project management. The “Project Management Body of Knowledge” (PMBOK) is often used as the master guide for managing a traditional project. In fact, when adapted properly, the PMBOK can be used to manage a project using any method, but that is a discussion for another time.

Sometimes people ask, “Which Agile method is the best?” People have a tendency to want the “best” of something. It is a valid question, and it has a clear and emphatic answer: “It all depends.” It all depends upon the nature of the project, the nature of management, the nature of the customer, the nature of the corporate culture, and the natures of the team members assigned to the project. It is that easy.

“The 10th Annual State of Agile Report” (the Report) is a survey that was conducted by VersionOne during CY 2015. According to the Report, the most common methods for implementing Agile are depicted in the following diagram, “Figure 1. Survey Results.”

The top five methods are Scrum[1], Scrum/XP Hybrid[2], Custom Hybrid, Scrumban[3], and Kanban[4]. These five methods constitute 88 percent of all the methods reported as being used. Each of the remaining methods is used by 3 percent or less of those surveyed. A beginner or intermediate user should follow the examples of others and use one of the preferred methods. A usage rate of 3 percent or less is deemed to be too low to warrant consideration by a beginner. Even 5 percent, 7 percent, or 8 percent may be rather low, but their inclusion brings the total domain being analyzed to 88 percent of the total domain of respondents. The diagram emphasizes the relative importance of each method.

Methods two through five total 30 percent of the methods used, which is slightly over half of the Scrum method. Possibly the opinions of many people, teams, and customers are sufficient for making a decision without reading further, but for those analytical thinkers and explorers, let’s consider further.

In many respects Scrum is easier to learn and execute than the other four methods, since two of them are hybrids that require some experience and the Scrumban method requires the blending of two methods (Scrum and Kanban).

Though Kanban is the fifth most popular method, features of Kanban are present in the Scrumban method. Scrumban incorporates certain aspects of Kanban, such as the Kanban board or establishing specific limits to the amount of Work in Progress (WIP). In Scrumban there is a relaxation of the goal of establishing a cross-functional team; specialization is permissible. Scrumban is

Figure 1. Survey Results
also easily usable for activities other than developing software.

Scrum is the most structured of the methods in that there are specific roles and specific rules.

- The specific roles are (a) the developers who produce the code, (b) the product owners who know what the product is supposed to do, (c) the scrum master who maintains the rhythm of development, and (d) an optional coach who helps a novice team or one that is seeking to improve.
- The specific rules are (a) to operate within specific time limits called “sprints,” (b) to plan tasks that can be completed within a sprint, (c) to work on only the most important tasks, (d) to hold a brief (15 minutes or less) meeting each day called the “daily stand-up” or “daily scrum,” during which only plans, progress, and problems are discussed, (e) to review and demonstrate the product to the customer at the end of each sprint, and (f) to review the successes and failures of the sprint in a group meeting called a “retrospective.”

Extreme Programming (XP) is based on the concept that “if some practice is good when developing software, then more is better.” Among the features of XP are constant code reviews, continual testing, continual integration, and anticipation of and handling of changes from the customer. Short time loops of all of these processes are promoted in XP.

The Scrum/XP Hybrid method can be considered as an extended form of Scrum because Scrum does not prohibit extensions. This method can include paired programming, continual integration, and, often, test-driven development. The Scrumban method may include these practices as well. Thus, there is a certain amount of overlap among the methods.

The Custom Hybrid method often means a hybrid of Scrum and Waterfall (the classical method for managing a project), but it can also refer to a hybrid of Scrum and some customization from the responding company. The survey does not provide details, nor does it require proof that the method claimed is even Agile, so the respondents who claimed to be using this method could be using a hybrid of Agile with something else or may only believe that they are using Agile without actually doing so. If the hybrid involves Agile and Waterfall, then it is likely that a contract or the customer requires such behavior (often contacts with a governmental agency). Hence, a hybrid method is needed, in which case a Custom Hybrid is the only method that can be used because it accommodates preliminary design, critical design reviews, and other milestones commonly found in governmental contracts.

Under such circumstances, no other method would be suitable; therefore, the best method is the Custom Hybrid method. Sometimes the Scrum-Waterfall method is called Scrumfall, and it can perform rather well. Training is required for skillful usage.

Kanban is a Japanese word meaning signboard or billboard. Its use was developed by the Toyota Motor Corporation. In the area of software, items to be worked are on a status board or “Kanban board.” Team members can view the progress of a task as it flows through to completion. Limits on Work in Progress (WIP) are imposed, though as a team develops a rhythm, the limits may be altered. Developing cross-functional skills is not a goal; specialization is the norm.

Team members “pull” work as capacity permits, rather than having work “pushed” into the queue of work. A growing backlog should be avoided. Flow should be smooth. The priority of work is established by its assignment in the queue, and effort is focused on items of higher importance. Work in Progress is controlled by limits defined by the team.

Scrumban contains most of the basic features of Scrum with some modification and the addition of some features from Kanban. Scrumban relaxes the goal of cross-functional teams; specialization is permitted. There is an enhanced emphasis on flow and queuing. Within a sprint, the product owner can change the priority of any task that has not already begun. More planning occurs when the backlog (or “to do” list) drops below a desired level to keep everyone busy.

So which practice is the best? One way to decide is to select the practice that is best suited to the team, the task, and the character of the company. Every company develops a culture, and some methods mesh better with certain cultures, so it may be valuable to try different methods. The phase of a project should also be considered. The development phase is different from the maintenance phase. Though the same team could perform either phase, the rhythm is different. New features come from the imagination; problems come from the real world.

More Considerations

Extreme Programming (XP) was once a very popular method for developing software, but the latest Report indicates that the use of XP has declined to 1 percent. XP has developed and

---

**Table 1. A Comparison of Roles and Practices**

<table>
<thead>
<tr>
<th>Practice</th>
<th>Scrum</th>
<th>Kanban</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-boxed iterations (sprints)</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>Team commitment</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>Cross-functional teams</td>
<td>Recommended</td>
<td>Optional</td>
</tr>
<tr>
<td>Task decomposed to fit within a sprint</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>Burndown charts</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>WIP limits</td>
<td>Defined by sprint length</td>
<td>Explicitly defined</td>
</tr>
<tr>
<td>Task estimation</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>Roles prescribed</td>
<td>Product Owner (PO), Scrum Master (SM), Developers (Dev.)</td>
<td>Optional</td>
</tr>
<tr>
<td>Backlog</td>
<td>Product and Sprint</td>
<td>Kanban board</td>
</tr>
<tr>
<td>Changing priorities</td>
<td>Not permitted during a sprint</td>
<td>Permitted if a task has not begun</td>
</tr>
</tbody>
</table>
fostered some practices that have been adopted by other methods, but the exclusive use of XP has declined.

Let us consider the set of Scrum practices and the set of Scrum/XP Hybrid practices to be components of a superset identified as “Enhanced Scrum.” Assuming that this superset exists, some form of Enhanced Scrum is used for 68 percent of projects. Hmm! Could this be sufficient evidence on which to base a decision? Again the answer is “It all depends.”

The constituent practices incorporated to form a company’s Custom Hybrid method are too variable to analyze, since the practices are tailored to meet the specific needs of an organization. This method should be reserved only for teams that are very skilled with using Agile.

Returning to the superset, Scrumban consists of a merger of Scrum and Kanban. How complete that merger is can be debated, but a true Agilist permits an Agile method to be flexible with the goal of being effective, so let’s include Scrumban in the superset of “Enhanced Scrum.” (Figure 2: Enhanced Scrum) Enhanced Scrum could contain all of the features of Scrum, portions of XP, portions of Kanban, and other practices that might prove beneficial.

Of the five listed methods, Scrum is the most structured method. Kanban is much less structured — possibly the least structured method. The other methods fall somewhere in between.

**Major differences between Scrum and Kanban**

Clearly, Scrum has many more roles and practices, making it more structured. Other methods are more flexible and have more optional practices. These aspects are both the strengths and weaknesses of each method. The weakness of an optional practice is that while it may be necessary for a certain team, it is often omitted because the team ignores optional practices. A frequent misinterpretation by new Agilists is that documentation is not needed. This is another discussion that will be ignored here due to complexity, but some documentation is mandatory; for example, user manuals, problem reports, user stories, and project status updates.

Teams new to Agile will probably be making a transition from a more structured environment. Therefore, the team will be more comfortable with a structured but flexible method. Psychologically, it will be easier to make the transition to Scrum than to the even more relaxed environment of Kanban. Also, project management professionals and the executives of a company will probably be more comfortable with Scrum than with Kanban. For these reasons, Scrum would probably be the better method for developing a new product or making major changes to an existing product. For teams skilled with Agile and Scrum, the transition from Scrum to Extended Scrum should be rather easy if it is desired or appropriate.

After a product moves to the operational phase, product support begins. Problems will arise and be reported, so Kanban may be the better method during this phase. Problems will establish the rhythm as they are reported. Each problem will be addressed for the time required, and the size of a problem will not be an issue. Each team member can select a problem on which to work, or the team can swarm onto solving a problem. Thus, during a project’s life cycle, different Agile methods may be appropriate.

**The Conclusions**

For a project involving mostly maintenance, Kanban would be a suitable method. Work focuses on each problem, irrespective of its size. An experienced team that does both development and maintenance might switch between Scrum and Kanban or use Scrumban.

The use of XP is fading, but many of its concepts are valuable, so they should be understood. An organization that examines XP may find it to be the most suitable method. However, the pool of experienced XP practitioners is smaller, so training will be almost mandatory.

So the answer to the question, “What Agile method is the best” remains clear: “It all depends upon what is to be done.” Has this helped?

---

**NOTES**


**REFERENCES**


**ABOUT THE AUTHOR**

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.

Agile Dev West Conference
4-9 June 2017
Las Vegas, NV
https://adcwest.techwell.com/

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

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

2017 12th International Conference on Computer Science and Education (ICCSE)
22-25 August 2017
Houston, TX
www.ieee-iccse.org

17th International Conference on Computational Science and Its Applications (ICCSA)
3-6 July 2017
Trieste, Italy
www.iccsa.org

26th International Conference on Computer Communication and Networks (ICCCN)
31 July – 3 August 2017
Vancouver, BC, Canada
www.icccn.org/icccn17

2017 39th Electrical Overstress Electrostatic Discharge Symposium (EOS/ESD)
10-14 September 2017
Tucson, AZ
www.esda.org/events/symposia

6th International Conference on Data Science, Technology and Applications (DATA)
24-26 July 2017
Madrid, Spain
http://www.dataconference.org/

16th International Conference on Cognitive Informatics & Cognitive Computing
26-28 July 2017
University of Oxford, UK
http://www.ucalgary.ca/icci_cc/iccc-17

Northrop Grumman Aerospace Systems Recruitment Events
2nd and 3rd Quarter 2017
Los Angeles, CA, San Diego, CA, Palmdale, CA, Oklahoma City, OK and Melbourne, FL
If interested, please send your resume to NGASrecruiting@ngc.com
A long, long time ago, in a country far, far away, there lived a Mighty King. The Mighty King was happy, as his computing needs were very simple. What simple programs he needed to run his kingdom could be punched up on less than 2000 punch cards. When the programs became obsolete, the King simply had the programs rewritten as needed. This method served the King's needs well.

And the King was happy.

But as computing power became greater, and the needs of the kingdom became more complex, the King had to change his method of program production. He summoned forth his wise consultants, and explained that the programs were becoming larger and more complex. He was at a loss at what to do.

And the King was unhappy.

The wise consultants of his kingdom explained to the benefits of an RFP. And the RFP was put forth. And responses were collected. And contracts were let. And requirements were gathered. And prototypes were developed. And the prototypes created great amounts of fertilizer.

And the King wept.

And JAD sessions were held, and requirements modified, and new versions of the software created. And this begat Version 1.

And the King was well pleased.

But the users began to complain, and asked for more and more features. And the King assembled a maintenance team together these additional requirements. But the requirements
were not prioritized. And the maintenance team tried to add every request into the second version of the system. And the days of coding were many, but unfortunately the days of testing were few. And even though there were still many errors, this effort prematurely brought forth Version 2. And it was slow. And it crashed frequently. And it had many features that very few people used. And it ate up all available memory. Yea, verily, did I mention it crashed frequently?

And the King waxed wroth.

And the King summoned his advisers, who explain to him about software bloat. And the King threatened to behead some developers. And lo and behold, a new design team for Version 3 was assembled. And requirements were re-gathered and re-verified. In the verified requirements were prioritized. And agile methodologies and object oriented design were used. In the days of analysis and design were great, and the days of coding were few. And because of the design process, and proper analysis and design, The days of testing were even fewer. And the system ran well, And the users were happy.

And the system performed as expected, and the King was well pleased.

And the days passed, and eventually the host hardware and the operating system were upgraded. And minor changes were made to the software, and it still ran well. For a while. And then more and more upgrades were required. Hardware changed again. The system was re-hosted to a different platform. And the changes grew and grew. And version 3.14 8.6g begin to crash with great regularity.

And the King was upset. His treasury was also bankrupt, and therefore he did not have the money to completely rewrite the software a fourth time

And the King summoned his advisers, who explained to him about re-factoring in maintenance. And the King was tutored in the areas of technical debt, and the magical spells for perfective, corrective, and adaptive maintenance. And as the system kept evolving, it worked less and less well. And the King endeavored to rewrite the system from scratch, but there was never enough time, and never enough money.

And the King was confused.

“Surely”, the King thought, “somebody in the kingdom can tell me when I need to rewrite the software, when maintenance no longer brings forth any benefits.” And many wise men were summoned and queried but no one had an answer that suited the king. Each contractor wanted their own company to do the rewrite, and each quoted an exorbitant price. And, the new federal oversight initiatives being what they were, not one contractor had the decency to offer the king a kickback.

Finally, after his advisers had given up, a very old sage was located, and summoned to appear before the king. This sage supposedly had the skill and expertise borne of many, many years of operations and maintenance experience, and even the wisest consultants considered him full of wisdom. He had truly earned the title of “graybeard”. At last he appeared, and went before the king and knelt. “How may I assist you” the sage asked. “Oh, great sage”, the King said, “Nobody seems to when to quit maintenance changes and rewrite the software again. I don’t even have an estimated cost. Do you have any idea when I can afford to rewrite the software again?”.

“Never”, said the wise sage. “It is never affordable. However, the day will come when you can’t afford not to!”

And the King saw the wisdom of these words, and realized that he had been doing the best that he could. The King immediately commissioned a task force to ask Congress for federal aid, and he also formed a committee to petition Congress to allow the new development money to fall under O&M funds.

And Congress agreed, and they all lived happily ever after.

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

*Note by author: I have wanted to use this phrase all my life. I heard it in the Groucho Marx movie Horse Feathers, made in 1932, well before my time. The lines were:

Secretary: “The Dean is furious. He’s waxing wroth.”

Quincy Adams Wagstaff [Groucho]: “Is Roth out there too? Tell Roth to wax the Dean for a while.”
CrossTalk thanks the above organizations for providing their support.