CrossTalk

May / June 2013 The Journal of Defense Software Engineering Vol. 26 No. 3

Large Scale Agile

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 2013		**00-00-2013 to 00-00-2013**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk, The Journal of Defense Software Engineering. Volume 26, Number 3. May/June 2013	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEB,6022 Fir Ave Bldg. 1238,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

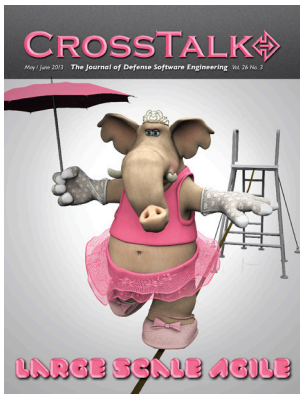
14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**36**	

TABLE OF CONTENTS

Departments



Cover Design by Kent Bingham

Large Scale Agile

CrossTalk

NAVAIR Jeff Schwalb
DHS Joe Jarzombek
309 SMXG Karl Rogers

Publisher Justin T. Hill
Advisor Kasey Thompson
Article Coordinator Lynne Wade
Managing Director Tracy Stauder
Managing Editor Brandon Ellis
Associate Editor Colin Kelly
Art Director Kevin Kiernan

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

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the U.S. Navy (USN); U.S. Air Force (USAF); and the U.S. Department of Homeland Defense (DHS). USN co-sponsor: Naval Air Systems Command. USAF co-sponsor: Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division in the National Protection and Program Directorate.

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

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

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the **CrossTalk** editorial board prior to publication. Please follow the Author Guidelines, available at <www.crosstalkonline.org/submission-guidelines>. **CrossTalk** does not pay for submissions. Published articles remain the property of the authors and may be submitted to other publications. Security agency releases, clearances, and public affairs office approvals are the sole responsibility of the authors and their organizations.

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

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

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

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

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

CrossTalk would like to thank NAVAIR for sponsoring this issue.

In the beginning, it was easy. Programming was fun. Requirements were clear and concise. Programs could be ripped out with pizza through an all-nighter. We were solo computer programmers. Then the group project appeared on the horizon. Most of us angled to go it alone. We did not need anyone else! We get stuck with two other people and do some rudimentary planning. Things do not go well and in the end, one person ends up doing 90% of the work because they do not trust the others to get anything done. Then we graduate from college and enter the real world where teams are the norm and solo work is almost non-existent. We enter a world for which we are wholly unprepared—the world of teams and the complexity it brings.

As I enter my 20th year as a professional software developer, more and more I see software projects like jigsaw puzzles in which the pieces are being shaped at the same time they are being assembled. If I am honest, software is probably more like a Rube Goldberg machine, but you get the idea. In fact, the slide in the DoD introductory acquisition training covering software project management has a drawing of the contraption from the board game Mousetrap, but I digress. Anyone who has assembled a large jigsaw puzzle knows the drill: look for the four corner pieces, find the edge pieces, start building the edges, look for color blocks, and so forth. The devilish thing about jigsaws is that they have exponentially rising difficulty relative to the number of pieces in the puzzle. Each piece has four sides, so if you have twice as many pieces, you have 16 times as many possible interconnects.

Software is similar to jigsaws in having an exponential number of interconnects but adds many more complications. The interfaces are flexible and ever changing. The functionality of the parts is ill defined. No one can lay all the pieces out on a large table for everyone to see all at once. Unlike jigsaws, and whether engineers (and their project managers) will admit it, engineering is a creative process. It exists in the minds of the creators until it is communicated in some way whether verbal or written. That is where the difficult project work begins.

Agile methods were originally developed to add two long-sought project attributes: short-term releasability and requirements churn flexibility. Agile thrives in an environment with high levels of verbal communication; the daily meetings; the on-hand stakeholders; and the pair programming. In small-scale agile, every member of the team knows what every other member of the team is doing. Cycles and tasks are short, and meetings are held often, so problems do not fester. One of the problems with agile methods are scaling up to large projects. You end up with teams of teams leading to groups of individuals not being on the same page. Programmers will know what their sub-team is working on in detail, but the other sub-teams' work will be more opaque. A common solution is documentation in the form of Interface Control Documents and formalized designs.

While the Agile Manifesto does not rule out internal project documentation, the creation of such does slow a project down and make it seem less agile. You end up with things like requirements sprints and architecture sprints before any usable product can be released to a customer. Should it be any surprise that a change in scale of a product would necessitate a change in the process used to create it? People get wrapped up in the pros and cons of one method versus another. Just because Scrum by-the-book does not fit your group does not mean that the Rational Unified Process is your only other choice.

I've been a Team Software Process (TSP) coach and I've been a ScrumMaster. At the moment, I am a TSP advocate but I'll be the first to admit that TSP is not the best method for every situation. It does not matter what you call your particular process. They all include planning, estimation, tracking, meetings, and, of course, writing software. Everything is tailorable. Dr. Deming's famous Plan-Do-Check-Act works perfectly here. Pick a process, use it, measure the results, and modify accordingly. Not one of the TSP teams I know of, and there are many, use TSP exactly by the book. They have all tailored the process according to their circumstances and metrics.

In the end, we are all just looking to get important work done on time, on budget, and with high quality. And maybe, just maybe, have some fun along the way.

Mark Stockmyer
Senior Computer Scientist
NAVAIR

A Lean Approach to Scheduling Systems Engineering Resources

Dr. Richard Turner, Stevens Institute

Abstract. Integrated Master Schedules serve a critical purpose in coordinating system development. However, in large operational systems where evolution is both rapid and externally driven, they can fail to provide the flexibility and visibility required for strategic decision making. Research into applying lean concepts to scheduling may hold the answer.

Understanding the status of evolutionary capability development in large operational systems is often difficult. Schedules are rarely stable due to:

- Size and complexity of capabilities
- Operationally imposed unexpected changes in priorities
- Deep supplier chains and contract structures
- Variety and availability of special engineering resources
- Generally complex nature of the operations.

This instability can cause undue stress on traditional scheduling models and tools. The effort required to maintain large networks and plans—exemplified by Integrated Master Schedules and Plans, complex work breakdown structures and earned value management systems—often outweighs their usefulness for communicating progress and managing resources. It is difficult to understand at any particular point in time:

- How much work the organization, program or project resources have the current capacity to perform within a specified time frame
- What resources are overcommitted or underutilized
- What work is actively proceeding
- How much work is actively proceeding
- What work is blocked and thus inactive
- What is the continuing impact of unpredicted changes in priority, urgent maintenance or critical development responses, changes in requirements or scope, or other emergent issues
- What is the actual progress toward various project outcomes
- How often and when is value finally delivered to the user

Lean approaches strive to maximize flow through a process—often by using on-demand (Kanban) scheduling techniques. Software development organizations have found that iterative and on-demand approaches are more flexible and provide better results than traditional push scheduling methods. The Systems Engineering Research Center (SERC), a University-affiliated research center of more than 20 universities and research organizations led by Stevens Institute, is investigating on-demand scheduling techniques in SE to determine if they can provide:

- Better status visibility managing multiple concurrent development projects
- More effective integration and use of scarce SE resources
- Increased project and enterprise value delivered earlier
- More flexibility while retaining predictability
- Less blocking of product team tasks waiting for SE response
- Lower governance overhead.

To investigate these aspects, SERC researchers have identified large, evolving, software-driven systems as the target environment for their initial work. These systems make up a significant portion of defense acquisitions, and include complex real-time actions that often occur across systems of systems.

After studying the needs of several government and commercial environments, the SERC team has combined several agile and lean ideas and is experimenting with their application in SE. We have defined a general Kanban-based Scheduling System (KSS) that we believe captures the essence of lean flow management visibility and flexibility. We have also developed a concept for SE as a service to support ongoing collaboration between SE and SW tasks. Together, we postulate that these approaches will improve the ability to reallocate resources as needed to meet emergent needs, continue to support ongoing development and maintenance activities, all without overloading resources. We are now describing and simulating the implementation of a network of KSSs in a complex, multi-site health care information system.

The KSS Concept

A KSS is a means of visually controlling workflow. It consists of a set of activities, where each activity has its own ready queue, a set of resources to add value to work units that flow through it, and a done queue.

Visual representation provides immediate understanding of the state of flow through the set of activities. This transparency makes resource issues and process anomalies (both common and special cause) easily visible, enabling the team to recognize and react immediately to resolve issues locally. Because the team and management interact with the visualization and collectively solve problems, this aspect is important in achieving continuous improvement (Kaizen). Control of the KSS is generally maintained through Work in Progress (WIP) limits, small batch size, and Classes-of-Service (CoS) definitions that prioritize work with respect to risk.

WIP is partially completed work, equivalent to the manufacturing concept of parts inventory waiting to be processed by a production step. WIP in knowledge work can be roughly associated to the number of work items started and not delivered. WIP Limits specifically cap the amount of work assigned to a set of resources. This lowers the context-switching overhead that impacts individuals or teams attempting to handle many simultaneous work items. WIP Limits accelerate useful value by completing work in progress before starting new work and also provide for reasonable and sustainable resource work loads.

CoS provides a variety of handling options for different types of work items and influence the next task selection within KSSs. They allow the WIP limits to be distributed in such a way that certain types of work will always take priority, will have more consistent access to resources, or will only be selected under certain circumstances.

The fundamental KSS building block is shown in Figure 1. In general, the upstream customer for the service provided is

responsible for selecting the work that enters the KSS. This is usually done collaboratively with the KSS to make sure that significant dependencies, date-certain events, and other special concerns are understood. As resources become available, the highest value work item is selected, resources are assigned, the work is executed until it is complete, and then added to the completed work queue. The value of a work item depends on a number of factors, including priority of the project associate with the work, cost of delaying the work, criticality of the work, and the work's impact across the larger system or system of systems. A scheduling cadence provides regular meetings of the KSS team to assess flow and determine if resources should be moved between activities, WIP limits adjusted, or other actions taken. Often, this is a daily activity, but the actual planning horizon selected and the nature of the work items should be used to establish the most cost-effective cadence.

SE as a Service

Defining SE as a service and using on-demand scheduling is designed to better allocate scarce SE resource and integrate the SE flow with the SW development project flow. If SE is seen as an overarching and somewhat separate activity, there is little ability to interact with the needs of the various development teams, and no means of identifying priority when asked for support. SE and developers both have unique insights into the rationale and reality of the systems or products they are evolving but often do not realize how their decisions impact their counterparts or the system as a whole. Viewing SE as a service performed for management or product/system developers generates communication opportunities that enable negotiation and collaboration in determining the priority, scheduling, and quality level of technical activities.

In general, SE is involved in three kinds of activities in rapid response environments: lifecycle, continuous, and requested. Lifecycle activities are critical in greenfield projects, but are important in all systems and system of systems evolution. They include front-end work like creating operational concepts, defining architectures, and capability and requirement decomposition and allocation, as well as final verification, validation, integration, and deployment activities. Continuous activities are ongoing, system–level activities (e.g. architecture analysis, performance analysis, configuration and risk management, and incremental verification, validation and integration). These require regular resources for analysis and the maintenance and evolution of long-term, persistent artifacts that support multiple projects. Requested activities are generally specific to either individual projects or capability engineering (e.g. issue triage, trade studies, impact assessments, needs analyses, cost analyses, interface support, and specialty engineering support), and draw on the persistent SE artifacts and knowledge.

By viewing persistent artifacts (architectures, requirements, interfaces specifications) as key components of services provided to various projects, SE can be opportunistic and apply its cross-project view and its understanding of the broader environment to better support specific projects individually or in groups. It can also broker information between individual projects where there may be contractual or access barriers. When a system-wide issue or external change occurs, SE can negotiate or unilaterally add or modify work items within affected projects to ensure that the broader issue is handled in an effective and compatible way. The quality of a service may be pre-specified,

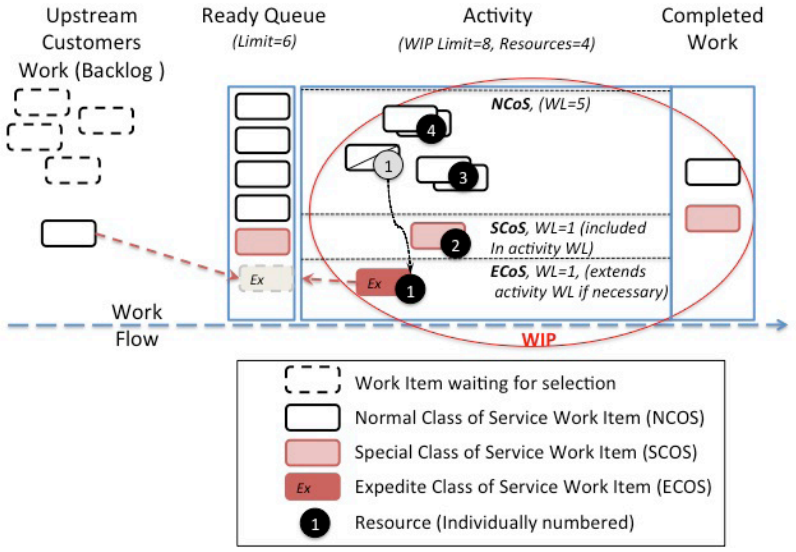


Figure 1. KSS Building Block

specified as a parameter of the service request, or negotiated as a function of typical value sought and time available to provide the service. SE services may be thought of as a single activity, although many activities are complex enough to have their own set of value-adding activities and specialized resources.

To support timeliness, SE performs its services in parallel to those activities in the requesting project. SE can use the KSS network constructs to compare the values of individual project work across the entire system and select the most critical work (often the work presenting the highest cost of delay) to accomplish next. This increases the effectiveness of the limited SE resources across the enterprise.

Implementing a KSS Network

The initial implementation uses a network of integrated KSSs that are intended to:

- Shorten the time required to deliver value to internal and external consumers
- Make work in progress and status visible at all levels through Dashboards and KSS flow boards
- Monitor organizational capacities at all levels
- Support analysis and decision making at every level of management
- Limit WIP to improve value flow (identify resource issues, cause of blocked work)
- Coordinate multiple levels of SE activity; enable cross-organizational teams and swarming of resources
- Establish a basis for continuous improvement in a rapidly changing environment

The KSS Network shows the relationships between the SW development tasks and the SE tasks. It also clearly captures the relationships between the SW and SE tasks and the capabilities. Understanding the information needs for decision making, including scheduling and flow monitoring/control, at each level of SE activity or utilization, is a key to a successful KSS design. Figure 2 shows the current conceptual design of the hospital system KSS Network.

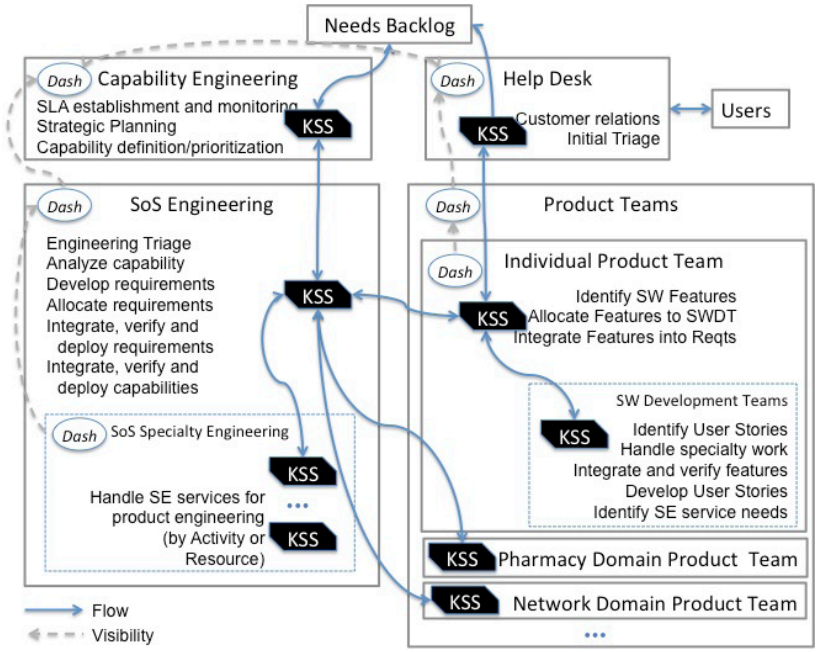


Figure 2. Healthcare LSS Network

Standard/enterprise-wide Classes of Service	
Critical Expedite	A Critical Expedite work item represents something that fixes an existing or imminent issue within the system. Safety, security, or other emergency work items are assigned this CoS. It is disruptive and requires all appropriately skilled resources to suspend their current activities and work on the Critical work item. It also suspends any WIP limits in activities associated with its work items for the duration that the critical work is in the activity. Once a work item is assigned this CoS, the CoS applies to all derived work items in all KSSs, regardless of local priorities.
Important	The Important CoS is assigned to very high priority work items where the speed of completion is such that this work should take priority over all other work in the ready queue. It is not disruptive, because all WIP is allowed to finish before the important work begins. It does not impact WIP limits, but has a guaranteed WIP limit in some KSSs.
Date Certain	Date certain (or schedule as independent variable) class of service reflects work items that must be completed by a specific date or there will be significant consequences. Regulatory implementation deadlines, COTS upgrade preparation, or integration/deployment dependencies are candidates for this class of service. It operates essentially like an Important CoS, but as the date becomes closer, it may elevate to Critical Expedite based on workload.
Standard	This is the normal CoS for the development organizations work. A high percentage of work should be assigned at this level for the KSS Network to provide the desired outcomes.
Background	Background work (sometimes referred as intrinsic or invisible) is work that must go on but is usually not time critical. It includes things like architectural enhancements, low-level technical debt, research and environmental scanning, or time-certain events not due in the near future. It is usually prioritized by its length of time in the queue (FIFO). Some KSSs may have a limit for the time background work can remain in the queue. When reached, this limit automatically triggers placement of the work item in a higher CoS.
Special Limited Classes of Service	
Collaborative	This special CoS is designed for activities that span organizations and organizational levels such as cross-discipline issue analysis. The actual mechanisms for implementing this CoS are not complete, but it will be designed so that shared resources can be tracked across multiple LSSs without changing the basic flow management and scheduling activities.
Product Support	This CoS is limited to certain SE LSSs that directly support Product Team requests. It is designed to limit the impact of other classes of service on work items resulting from product teams requests. There is a guaranteed WIP Limit allocation for this work, meaning there are always resources allocated to this class of work, preventing the complete blockage of flow. It also raises the priority or value of work over time similarly to the Background CoS.

Table 1. Initial CoS for Health Care Systems KSS Network

The KSS Network acts as a distributed database of changing work status and value. This provides a basis for informed decision making at every level and encourages pushing technical decisions to the lowest level appropriate. A critical characteristic is the transparency provided by the near-universal status availability and the specificity of the policies that underlay the scheduling decisions. These policies are most often defined using CoS, WIP limits, and value definitions, and are exercised by informed, collaborative decision makers.

The CoS that have been identified for the heath care system KSS Network are presented in Table 1. The definition of initial WIP Limits, collaboration mechanisms for specific types of work items, and value determination algorithms are still underway.

Conclusions

This research has included a great deal of wandering through exciting possibilities and running into stark realities. The concepts are reasonable, but the work in applying them to the difficult environments we have chosen is just beginning. The next step is to simulate the example health care LSS Network and experiment with the mechanisms we have defined to implement its activities. This is not a simple task.

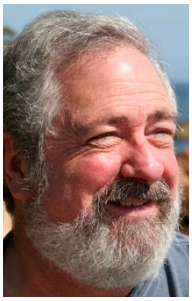
To support both the concept evolution and the simulation development, we are also looking for interested organizations or projects to pilot single or multiple level LSS concepts on their own work flow. The data gathered and experience provided by such pilots would be extremely helpful to the research, but may also support the pilot organizations in beginning the culture change initiatives that inevitably accompany transition.

Acknowledgements:

This material is based upon work supported, in whole or in part, by the DoD through SERC under Contract H98230-08-D-0171. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology. The SERC RT-35 Research Team includes Richard Turner, PI, Stevens Institute; Ray Madachy, Co-PI, Naval Postgraduate School; and Barry Boehm, Jo Ann Lane, and Dan Ingold, University of Southern California.

Additionally, a volunteer Industry Working Group has been essential to formulating the approaches with an eye to the needs of the work environment. Its members include David Anderson (David J. Anderson and Associates), Jabe Bloom (The Library Corporation), Hillel Glazer (Entinex), Curtis Hibbs (Boeing), Suzette Johnson (Northrop Grumman), Larry Maccherone (Rally Development), Don Reinertsen (Reinertsen & Associates), David Rico (Boeing), Garry Roedler (Lockheed Martin), Karl Scotland (Rally Software, UK), Alan Shalloway (NetObjectives), Neil Shirk (Lockheed Martin), Neil Siegel (Northrop Grumman), and James Sutton (Jubata Group).

ABOUT THE AUTHOR



Dr. Richard Turner has more than thirty years of experience in systems, software and acquisition engineering. Currently a Distinguished Service Professor at the Stevens Institute of Technology in Hoboken, New Jersey, he is co-author of three books: Balancing Agility and Discipline: A Guide for the Perplexed, co-written with Barry Boehm, CMMI Distilled, and CMMI Survival Guide: Just Enough Process Improvement. Dr. Turner is a Fellow of the Lean Systems Society.

E-mail: rturner@stevens.edu

RECOMMENDED READING

1. Boehm, Barry and Turner, Richard (2004). Balancing Agility and Discipline: A Guide for the Perplexed. Boston, MA: Addison Wesley.
2. Larman C. and Vodde, B. (2009). Scaling Lean & Agile Development. Boston, MA: Addison Wesley.
3. Poppendiek, Mary. (2007). Implementing Lean Software Development: Boston, MA: Addison Wesley.
4. Turner, Richard and Wade, J. (2011). Lean Systems Engineering within System Design Activities, Proceedings of the 3rd Lean System and Software Conference, May 2-6, 2011, Los Angeles, CA.
5. NDIA-National Defense Industrial Association (2010). Top Systems Engineering Issues In US Defense Industry. Systems Engineering Division Task Group Report, <http://www.ndia.org/Divisions/Divisions/SystemsEngineering/Documents/Studies/Top%20SE%20Issues%202010%20Report%20v11%20FINAL.pdf>, September, 2010.
6. Turner, Richard, Shull F., et al (2009a) "Evaluation of Systems Engineering Methods, Processes and Tools on Department of Defense and Intelligence Community Programs: Phase 1 Final Technical Report," Systems Engineering Research Center, SERC-2009-TR002, September 2009.
7. Turner, Richard, Shull F., et al (2009b) "Evaluation of Systems Engineering Methods, Processes and Tools on Department of Defense and Intelligence Community Programs: Phase 2 Final Technical Report," Systems Engineering Research Center, SERC-2010-TR004, December 2009.
8. Anderson, David. (2010). Kanban: Successful Evolutionary Change for Your Technology Business. Sequim, WA: Blue Hole Press
9. Burrows, Mike. (2010). Kanban in a Nutshell. Blog post. <http://positiveincline.com/index.php/2010/03/kanban-in-a-nutshell/>, March, 2010.
10. Reinertsen, Donald G. (2010). The Principles of Product Development Flow. Redondo Beach, CA: Celeritas Publishing.
11. Boehm, B. et al. "Applying the Incremental Commitment Model to Brownfield Systems Development," Proceedings, CSER 2009, April 2009.
12. Boehm, B., and Lane, J., "Using the ICSM to Integrate System Acquisition, SE, and Software Engineering," CrossTalk, October 2007, pp. 4-9.
13. Turner, R., Madachy R., Ingold D., and Lane J., "Modeling Kanban Processes in Systems Engineering," submitted to International Conference on Software and System Process 2012, 2012.
14. Turner, R., J.A. Lane, D. Ingold, R. Madachy, and D Anderson. "An event-driven, value-based, pull systems engineering scheduling approach." Systems Conference (SysCon), 2012 IEEE International,. IEEE, 2012. 1-7.
15. Office of the Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering (2008). Systems Engineering Guide for Systems of Systems, Version 1.0. Washington, DC: ODUSD(A&T)SSE, 2008.

Scaling Agile Development

Large and Multisite Product Development with Large-Scale Scrum

Craig Larman, Bas Vodde

Abstract. Since 2005 we have worked with clients to apply the Scrum framework and to help scale agile development to product groups involving from a few hundred to a few thousand people, in multiple sites. This is organized as large-scale Scrum frameworks 1 and 2, summarized in this article and elaborated on in our two-volume book series on very large-scale agile development.

Background

In 2003, when Craig Larman published Agile & Iterative Development [1], many "knew" that agile development was for small groups. However, we became interested in—and got increasing requests—to apply Scrum to very large, multisite, and offshore product development. So, since 2005 we have worked with clients to scale up—often for "embedded" systems. Today, the two large-scale Scrum frameworks described herein have been introduced to big groups worldwide in disparate domains, including telecom-infrastructure-equipment providers such as Ericsson [2], and investment-banking clients such as Bank of America-Merrill Lynch, plus many more.

To quantify "large", we have seen our large-scale Scrum framework-2 applied in groups of up to 1,500 people, involving seven developments sites spanning the globe. Our median experience is perhaps around 800 people on one product at 5 sites, with about 15 million lines of source code, usually C++, C, and Java.

Based on these experiences we published two volumes on scaling agile development and the large-scale Scrum frameworks summarized: (volume 1) Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum [3], that explains the leadership and organizational design changes, and (volume 2) Practices for Scaling Lean & Agile Development: Large, Multisite & Offshore Product Development with Large-Scale Scrum [4], that explains concrete suggestions for scaling, including in product management, architecture, planning, multisite, offshore, and contracting. This article summarizes concepts expanded on in those books.

Large-Scale Scrum is Scrum: Change Implications

Scaling Scrum starts with understanding and being able to adopt standard real one-team Scrum. Large-scale Scrum requires examining the purpose of single-team Scrum elements and figuring out how to reach the same purpose while staying within the constraints of the standard "Scrum rules."

If the entire R&D group was only seven people, the implications of changing to adopt true one-team Scrum are not dramatic, since many elements will "organically" be in place—as in a startup. But when a traditional R&D group of 500 people moves to Scrum, there are major change implications, and these need full understanding and support by senior leadership and hands-on producers. These include:

1) Standard Scrum: A small (five to nine people) cross-functional team of multi-learning team members that do everything end-to-end to develop the product (a real feature team [5]), and no specialized sub-groups within the team, with the only title of "team member" [6].

Change/scaling implications:

- No separate analysis group, testing group, architecture group, user experience group, platform group, etc. And no "tester" or "architect" within the team. That implies the dissolution of existing single-function groups and the management supervising roles, and the elimination of traditional career paths and job titles.

2) Standard Scrum: The business-person "owner of the product" (such as, lead product manager) responsible for ROI and cost, and who can independently decide and change the content and release date becomes the Scrum product owner. The owner of the product steers development directly based on "inspect and adapt" and so is ultimately responsible for the product release, since they have the steering wheel.

Change/scaling implications:

- Traditionally, the owner of the product negotiated a scope-and-date milestone-based internal contract with R&D managers, who were thereafter responsible for the release. Since the owner of the product now steers directly, there is no shifting of responsibility to R&D to develop the release, and no internal contract.
- Since the owner of the product steers development directly and is responsible for the release, there is no separate R&D or IT program/project manager responsible for the release; that role is eliminated.

3) Standard Scrum: Each two- to four-week Sprint, from the first, the product increment must be done and potentially shippable— a potentially shippable product increment. Each Sprint the system must be implemented, integrated, fully tested, documented, and capable to deploy.

Change/scaling implications:

- The concept of a "big release" and the constraint, "it is not ready until the end" dissolves. This implies eliminating big release management systems, practices, roles, and policies that are predicated on a long phase of messy partially-done development before the system is ready.
- Scrum is not for the programming phase after analysis and before testing. There is no prior analysis phase or architecture phase and no following integration/testing phase. Sequential lifecycle development is eliminated, and with it, the groups that were attached to each phase (the analysis group, ...).

4) Standard Scrum: The team is self-organizing (self-managing) and is empowered to independently decide how to achieve their goal in the Sprint.

Change/scaling implications:

- There is no team lead or project manager that directs or tracks team members, which implies the elimination of related team-lead and project-manager roles.
- No organization-wide standard process that everyone must follow.

This is simply standard one-team Scrum, but its adoption especially challenges traditional R&D assumptions and organizational design at scale. Therefore, most groups do not adopt real Scrum, but instead customize it (into "fake Scrum" or "Scrum, but…") rather than change themselves.

Observations and suggestions:

- Real agile development with Scrum implies a deep change to become an agile organization; it is not a practice, it is an organizational design framework.
- Start a large-scale agile Scrum adoption by ensuring leadership understands the organizational implications, and they have been proven adoptable in the small scale.

Two Agile Scaling Frameworks

After the aforementioned organizational-design changes are understood by leadership and they flip the system, then one of two large-scale Scrum frameworks can be adopted. Most of the scaling elements are focusing the attention of all the teams to the whole product instead of "my part". Global and "end-to-end" focus is perhaps the dominant problem to solve in scaling. The two frameworks – which are basically single-team Scrum scaled up – are:

- Framework-1: Up to 10 Scrum teams (of seven people).
- Framework-2: Up to a few thousand people on one product.

Framework-1 is appropriate for one (overall) Product Owner (PO) and up to 10 teams. 10 is not a magic number for choosing between framework-1 and framework-2. The tipping point is context dependent; sometimes less. At some point, (1) the PO can no longer grasp an overview of the entire product, (2) the PO can no longer effectively interact with the teams, (3) the PO cannot balance an external and internal focus, and (4) the product backlog is so large that it becomes difficult for one person to work with. When the PO is no longer able to focus on high-level product management, something should change.

A group with seasoned people who know the product and customers well and are co-located with the PO can handle more teams with one PO. A newly formed outsourcing group in India who do not know the domain, with a PO in Boston, will require less teams.

Before switching to framework-2, first consider if the overburdened PO can be helped by delegating more work to the teams. Encourage teams to directly interact with real customers to reduce handoff and reduce the burden on the PO. Most detailed analysis and project management should be done by the teams; the PO does not need to be involved in low-level details – they should be able to focus on true product management.

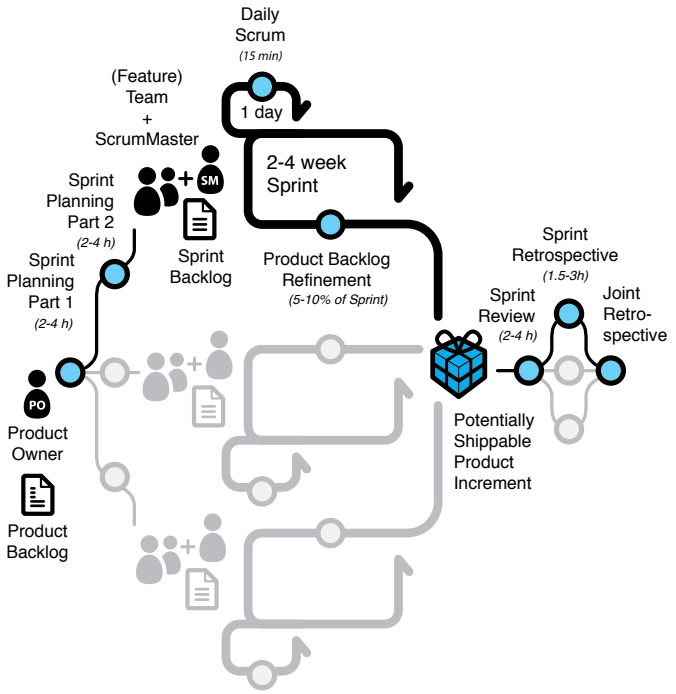


Figure-1. Large-scale Scrum framework-1

What is the Same as One-Team Scrum?

- One product backlog (it is for a product, not a team)
- One definition of done
- One potentially shippable product increment
- One (overall) product owner
- Each team is a "team" (and there are no single-specialist teams)
- One Sprint

In summary, all teams are in a common Sprint to deliver a common potentially shippable product increment.

What is Different?

- Role changes: none.
- Artifact changes: none; but to clarify: Sprint backlog and Sprint goal per team.
- Meeting changes: The dominant difference in large-scale Scrum framework-1 is the behavior of Scrum meetings, driven by coordination needs. This is illustrated in Figure-1 and explained next:

1. Sprint Planning Part 1: One meeting, and same maximum duration: one hour per week of Sprint. Rather than all team members participating, limit it to two members per team, plus the one overall PO. Let team representatives self-manage to decide their division of product backlog Items, although if "competition" exists the PO can break a tie. End with the partial-teams identifying dependencies (perhaps with a dependency matrix) between PBIs and discussing coordination.

2. Sprint Planning Part 2: Independently (and usually parallel) per team, though sometimes a member of team-A may observe team-B's meeting and make suggestions when there is a coordination issue between the teams.

3. Daily Scrum: Independently per team, though a member of team-A may observe team-B's Daily Scrum, to increase information sharing.

4. **(addition) Inter-team coordination meeting:** Several times per week, team representatives may hold an open space, town hall meeting, or Scrum of Scrums, to increase information sharing and coordination.

5. **(addition) Joint light product backlog refinement:** Maximum duration: 5% of Sprint. Only two representatives per team. Splitting, analysis, and estimation for soon-to-develop PBIs. Analysis is lightweight; for example, if using Specification by Example, only three examples per item. Note that the cross-team estimation ensures a common baseline for estimation across teams. Note that this meeting increases product-level learning and team agility – the ability of any team to take on any PBI.

6. **Product Backlog Refinement:** For this mid-Sprint meeting preparing for future Sprints, for co-located teams, hold this at the same time in one big room with all team members; each team facing a separate wall with their own learning tools (whiteboards, projectors, ...). Apply rotation writing and other large-group workshop techniques so that all members across all teams are eventually exposed to analyzing all items, which is critical for more team flexibility.

7. **(optional addition) In-Sprint PBI Inspection:** When possible, informally seek out early feedback from the PO or other stakeholders on finished PBIs as soon as possible during the Sprint, to reduce the inspection and discussion that would otherwise be required at the Sprint review; this does not eliminate the Sprint review.

8. **Sprint Review:** One meeting, and same maximum duration: one hour per week of Sprint. Limit it to two members per team, plus the PO and other stakeholders. Rather than only a common inspection of the running potentially shippable product increment, consider a "bazaar" or "science fair"-style phase during the middle of the Review: a large room has multiple areas with computers, each staffed by team representatives, where the features developed by a team are shown and discussed. In parallel, stakeholders visit areas of interest and team members record their feedback. However, begin and end the Sprint Review with everyone in a common discussion, to increase overall feedback and alignment.

9. **Team Retrospective:** Independently per team; no change.

10. **(addition) Joint Retrospective: Maximum duration:** 45 minutes per week of Sprint. Since the team retrospective ends the Sprint, this Joint Retrospective is held early in the first week of the subsequent Sprint. ScrumMasters and one representative of each team meet to identify and plan improvement experiments for the overall product or organization.

Agility Across Teams

Notice that large-scale Scrum increases learning across teams; most can flexibly do any Product Backlog item. This is in contrast to "team A can only do A-type work", and critical for agility when scaling, so that teams are responsive to change, and all can focus on the highest-value work, rather than constrained by single specialty. Remember: agile development is for agility (flexibility) over efficiency.

Coordination

When scaling, a dominant issue is coordination. In traditional scaling, this is (poorly) handled with major upfront "fixed" specifications and planning, private-code component teams, and extra managers. In scaling agile development, coordination is handled more by increased coordination in shared code and self-organizing teams. Besides meetings, what other coordination elements are in large-scale Scrum?

- **Continuous integration:** All code, across all teams, is integrated continuously (many times per day) and verified with automated tests, with a "stop and fix" culture of rapidly fixing a broken build.
- **Internal open source:** Rather than private-code components and "component teams", there is collective code ownership or "internal open source." Many open-source practices apply, such as standard coding style.
- **Feature teams:** Scrum feature teams develop end-to-end "vertical" customer-centric features across all shared code.
- **Communities of practice (CoPs):** To handle cross-team concerns (architecture, user experience, standards, ...) CoPs are established (and all that implies), with membership from the Scrum teams (not from external people). For example, a Design/Architecture CoP for the key concern of good design at scale; this is not composed of a separate "architecture group", but by volunteering regular Scrum team members with the skill and passion.
- **Team-controlled build system:** Rather than a separate "build group", regular Scrum teams rotate responsibility for maintaining their common build system.
- **More talking!**

Notice as a theme that coordination is handled by self-organizing teams (rather than more managers), and with fast-feedback integration cycles in code (rather than more planning and separated code).

Multisite

If an entire product group is seven people in four sites then a co-located team is difficult. But when 50 people, it is possible to create co-located teams of five to nine people: three teams in Boston, etc. Therefore:

- **Co-located teams:** Although different co-located teams may be in different sites, avoid a single dispersed team with scattered members. The motivation for dispersion is usually specialist bottlenecks ("only Mary knows X") but a key value in Scrum is to increase learning and multi-skill to reduce bottlenecks, rather than accept them.
- **Continuous integration across all sites:** And related...
- **Free open-source (FOSS) tools:** Especially when multisite, we observe frictions in groups using commercial tools... "We cannot have more licenses", "Wait for purchasing" etc. FOSS tools (Subversion, Git, GNU tools, Eclipse, Java, etc.) eliminate friction, reduce costs, and are usually superior.
- **Free "Web 2.0" information tools:** Multisite requires more software tools; use FOSS wikis, Google Docs, and other free "pure Web" tools for information (lists, requirements, etc.), rather than commercial and pre-Web document-based tools such as Word, SharePoint, DOORS.

• Free ubiquitous video: Rapport and trust—critical! And it is degraded when people do not see each other, so replace phone calls with video. Use free, ubiquitous tools such as Google Video Hangouts and a projector.

• Multisite Sprint Planning Part 1: How? The PO is with local representatives. Other sites use video and web tools. The PO offers items via a web tool (e.g., Google Spreadsheet). Parallel discussion on the items happens on different wiki pages or chat sessions.

• Multisite Product Backlog Refinement: As in Planning Part 1, emphasizes video and web tools. If estimating with Planning Poker, use (for example) a Google Spreadsheet with different members typing estimates into different cells.

• Multisite Sprint Review: As above.

• Multisite communications CoP: Good communication requires meta-communication.

Product Backlog	
Backlog Item	**Requirement Area**
IPv6	protocols
performance 10x	performance
HSDPA	management
performance stats	protocols
configuration of cells	management
new NMS solution	continuous integration
speed-up of build	upgrades
improved upgrading support	management
stability to 99.999%	reliability

Figure-2. Requirement Areas

Requirement Areas

With 1,000 people on one product, divide-and-conquer is unavoidable. Traditional development divides into single-function groups (analysis, ...) and architectural-component groups (UI-layer group, ...), yielding slow inflexible development with high levels of waste (inventory, work-in-progress, handoff), long-delayed ROI, and weak feedback. And it is organized "inward" around function and architecture, rather than "outward" around customer features.

In large-scale Scrum framework-2, we do not divide by architecture; rather, we divide around major areas of customer requirements – requirement areas. For example: fault management or options trading. Then, we add a "requirement area" column to the Product Backlog and classify each item in one area (Figure-2). A filter on one Product Backlog shows distinct Area Backlog views (Figure-3).

New Role: (Requirement) Area Product Owner

To deal with the overwhelming complexity for one PO, we introduce a new role: an area PO, who focuses on one area backlog.

The one overall PO plus all area POs form the product owner team.

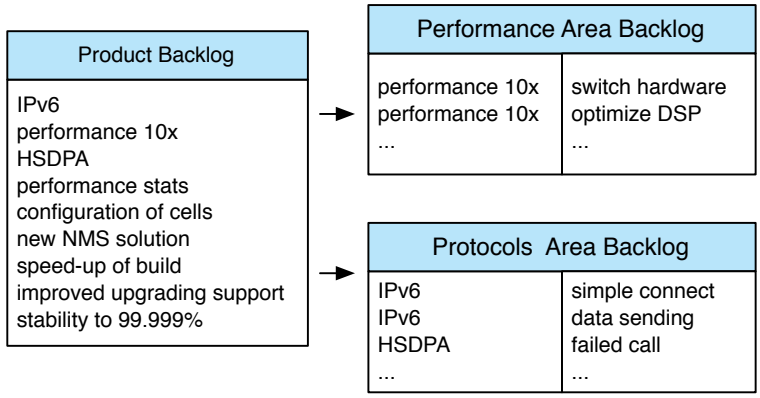


Figure-3. (Requirement) Area Backlogs

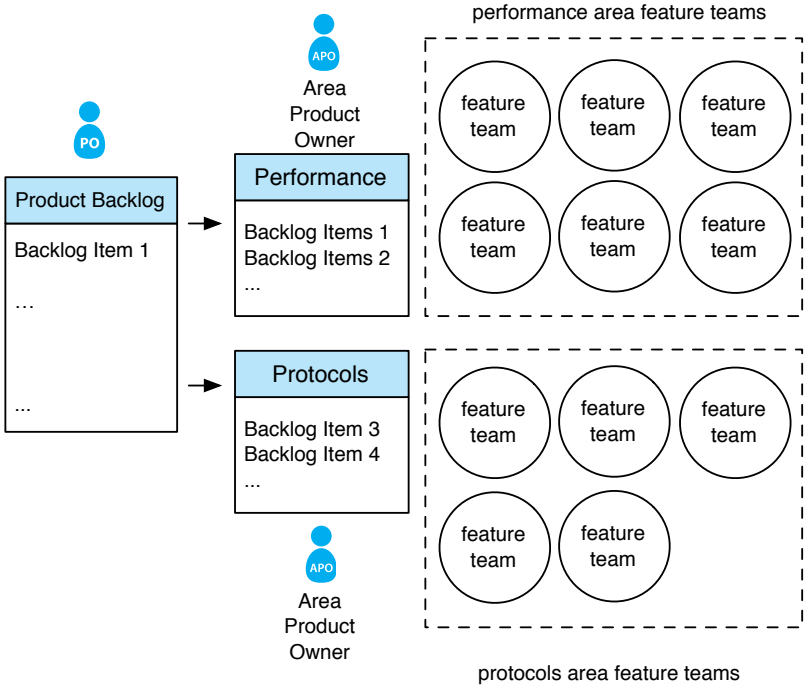


Figure-4. Area PO and Teams

Area Teams

A set of three to 10 teams (area teams) are dedicated to one area PO, all who specialize in one requirement area (Figure-4). Each team is cross-functional and cross-component, doing end-to-end customer-centric feature development.

The Big Idea?

Large-scale Scrum framework-2 is a set of several framework-1 groups (one per requirement area) working in parallel in a common Sprint (Figure-5).

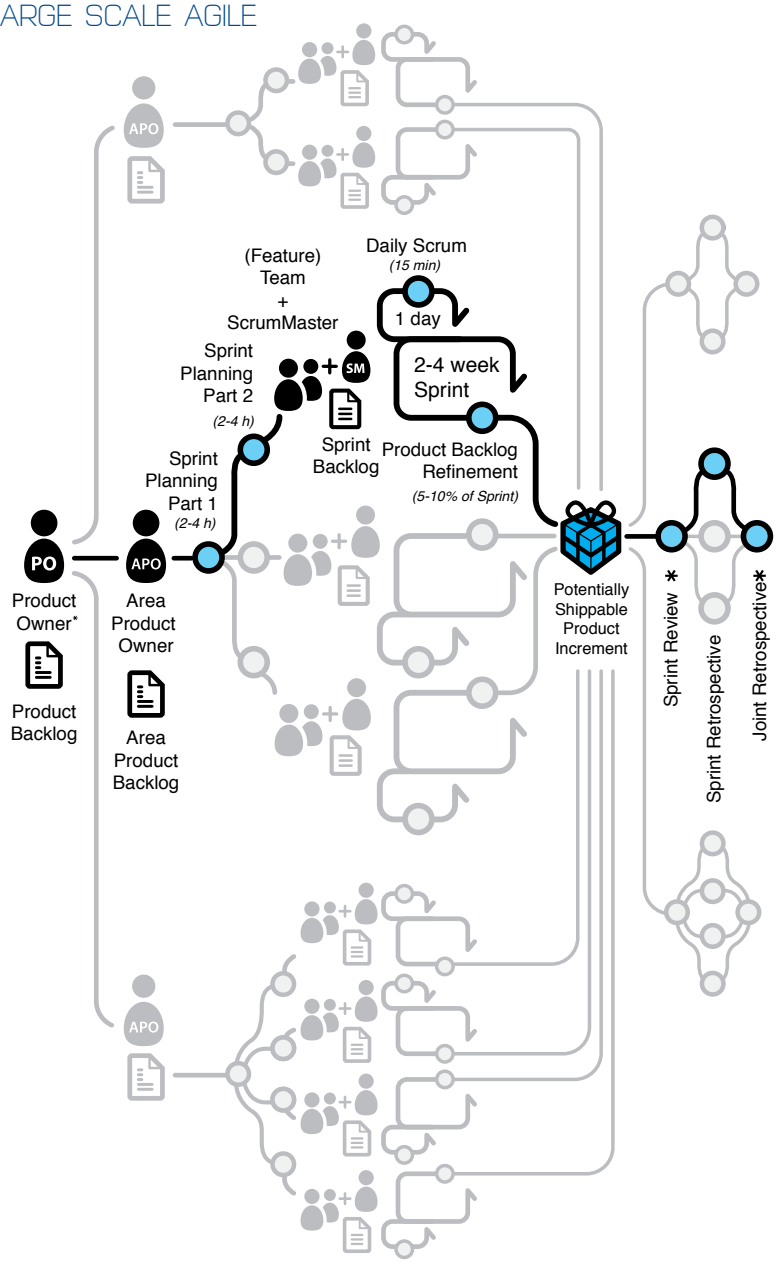


Figure-5. Large-Scale Scrum Framework-2

What is the Same as Framework-1?

One product backlog, one definition of done, one potentially shippable product increment, one (overall) product owner, one Sprint. All teams in one Sprint with one delivery.

What is Different?

- Role changes: area PO.
- Artifact changes: "Requirement areas" in product backlog; area backlog views.
- Meeting changes: Framework-2 is a set of parallel (per requirement area) framework-1 Sprint executions; therefore ...

1. Pre-Sprint Product Owner Team Meeting: Before each area PO meets in their own Sprint planning Part-1 meeting with their area Teams, they need to coordinate together and with overall PO – who focuses on product-level rather than area-level optimization. This coordination must happen before the Part-1 meetings, usually late in the prior Sprint.

2. Area-Level Meetings: As in normal framework-1, Sprint planning part 1, joint product backlog refinement, Sprint review, and joint retrospective need to occur for each requirement area.

3. Overall Sprint Review: A review is needed at the product level, not merely each area. It is not possible to review all items of all areas, so the focus is on a subset of interest to the overall PO or to many area POs.

4. Overall Sprint Retrospective: For system-level improvement, a retrospective is needed at the product level, not merely each area. This happens earlier in the subsequent Sprint, after area-level joint retrospectives.

ABOUT THE AUTHORS

Craig Larman is an organizational-design consultant for enterprise adoptions and very large-scale product development with large-scale Scrum, especially in embedded systems domains, and investment banking. He serves as chief scientist at Valtech, and is the co-author (with Bas Vodde) of Scaling Lean & Agile Development: Thinking & Organizational Tools and Practices for Scaling Lean & Agile Development: Large, Multisite, & Offshore Product Development with Large-Scale Scrum, and Agile & Iterative Development.

E-mail: craig@craiglarman.com
Website: www.craiglarman.com

Bas Vodde is a coach, consultant, trainer, and author in modern agile and lean product development. When coaching, he works on three levels: organizational, team, individual/technical practices. He has trained thousands of people in software development, Scrum, and agile practices for over a decade. He is the co-author of two books on large-scale Scrum and agile development, with Craig Larman. Bas works for Odd-e, which supports organization in improving product development, mainly in Asia.

E-mail: basv@odd-e.com
Website: www.odd-e.com

REFERENCES

1. Larman, Craig. Agile & Iterative Development: A Manager's Guide. Addison-Wesley, 2003.
2. Ericsson R&D Center Finland, "How we learn to stop worrying and live with the uncertainties". <https://www.cloudsoftwareprogram.org/results/deliverables-and-other-reports/i/27891/1941/ericsson-journey-of-change>
3. Larman, Craig & Vodde, Bas. Scaling Lean & Agile Development: Thinking & Organizational Tools for Large-Scale Scrum. Addison-Wesley, 2008.
4. Larman, Craig & Vodde, Bas. Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum. Addison-Wesley, 2010.
5. Larman, Craig & Vodde, Bas. Feature Team Primer. <http://www.featureteamprimer.org/>
6. Vodde, Bas. "Specialization and Generalization in Teams". <http://www.scrumalliance.org/articles/324-specialization-and-generalization-in-teams>

Uncomfortable With Agile

Andy Hunt, The Pragmatic Programmers, LLC

Abstract. The original intent behind the agile software development movement and its goals have become diluted over the past decade, with many teams simply doing selected agile practices instead of becoming agile practitioners. Many adopters and their organizations now claim to be comfortable with the idea of agile software development and their own use of it, which is probably an indication that they are doing it wrong.

Introduction

The Agile Manifesto was created more than 10 years ago, in February 2001, by a group of 17 interested individuals. I was fortunate to be one of those 17; witnessing and participating in the articulation and advancement of an idea that would, we hoped, change the face of software development for the better. But 10 years is a long time in the real world, and it is an eternity in the tech world.

As a result, things did not quite turn out the way we expected. In 2001, when we created the manifesto and launched the agile software development movement, we fully expected to see a crowded rush of new methods, new practices, and new ways of being agile. These new ideas would follow the guidelines of the manifesto but introduce new ideas and approaches and help advance the art. That did not happen.

Instead, we have seen wider-scale adoption of individual practices that agile software development favors, including formerly controversial practices such as pair-programming, and even basic hygienic practices such as version control that previously were not always followed. Overall, this seems to have had a very positive effect on many development organizations, but it is surely less than many of us had hoped for.

It seems that the initial focus of the agile software development has been forgotten over time, and the goal has shifted from becoming adaptive, flexible, agile practitioners to merely following a subset of canonical, prescribed agile practices. It appears that people have forgotten why we do what we do.

Of course it is not just software development that suffers from this sort of phenomena. For instance, when Osama bin Laden was caught and killed in May of 2011, there was a huge spike in search traffic asking, "who is Osama bin Laden?" Some two-thirds of the searchers were teenagers, which casts serious doubt on our ability to transfer information to the next generation [1]. We, as a people, had forgotten why we were doing what we were doing.

If our collective mass memory can forget such significant, pivotal events in our shared global history, then remembering the subtleties of a niche topic like "effective ways to develop software" is probably doomed from the outset.

So it is not too surprising to consider that perhaps the original intent behind the agile software development movement and its goals have become diluted over time, and perhaps have not been transferred to our peers, newcomers to our teams, or our successors. It is not surprising that many adopters and their organizations now claim to be "comfortable" with the idea of agile software development and their own use of it.

If you consider yourself comfortable with "agile," then that is too bad, because it probably means you are doing it wrong.

Uncomfortable With Agile

Software project teams seem to show a tendency toward one of two extreme states:

1. A comfortable, repetitive routine (some may say "somnambulant") that ends in disaster when the team is blindsided by new developments (be they from the market, the technology, or the organization).

2. Chaotic disarray, where the main driving force is sheer panic, fueled by unrelenting overtime, conflicting goals and priorities, and enough poor decisions to fuel a Dilbert calendar.

Given the choice, however, you do not want to just aim for the middle of these cartoonish extremes.

Instead, you want to creep up to the very edge of that terrifying, wild mess of dysfunctional development disaster: up to the edge of the abyss, but without falling in. Think of that uncomfortable feeling of tipping back in your seat, just before you lose your balance and catch yourself. That is where an agile practice needs to be in order to thrive—balanced on a narrow edge that requires constant small adjustments to maintain equilibrium.

I think the best definition I have seen that captures this spirit of agile comes from Dr. Patricia Benner, author of From Novice to Expert. Speaking on the nature of expertise and how to train people in real-world practices (clinical nursing in this case), she said, "Practices can never be completely objectified or formalized because they must ever be worked out anew in particular relationships and in real time." [2]

That is, you can never completely define agile, or its practices, because they are constantly evolving to meet specific needs in specific circumstances. Agile should be ever-shifting, ever-changing, ever-responding to change in context. As a practitioner of agile software development, you need to keep thinking, and keep adjusting.

I humbly suggest that this fundamental idea of agile development is what we (collectively) have forgotten.

What Is Agile?

In the crush and press of the exigencies of the day, we have forgotten some of the theoretical underpinnings of agile methods—why we do what we do. I would say agile includes ideas from:

- **Chaos Theory**
- **Kaizen**
- **Systems Thinking**
- **Risk Management**
- **Return on Investment**

Some of the key ideas from chaos theory include the extension of the pure physics version to organizations: that people, teams, and software products are inherently non-linear systems. That means that cause-and-effect can be impossible to isolate and identify. That is not a flaw in your team or project, it is merely a fact of life.

So is emergence, that nearly-magical phenomenon where complex systems and patterns spontaneously emerge from simple interactions. The converse seems true as well, that complex interactions tend to create simplistic, even "stupid", behavior. This phenomenon is a large part of the agile drive to try and keep the interactions and rules of development simple, clear, and low-ceremony.

If you are an agile practitioner, do you regularly observe complex behavior arising from your code or your team? Or just the opposite?

Then of course there are the well-known ideas from Kaizen, emphasizing using quick feedback to make continuous changes, and so to create an environment of continuous improvement.

Are you using constant feedback to make continuous changes to your product and process?

From Systems Thinking, we came to the idea of people as First-Order Components of the systems we are building. We are not resources, we are not staffing, we are people, and we are as much a part of the system as the database is. We realized that it was up to us as individuals; that particular practices, special tools, or novel processes would not save us.

So are you investing in the people on your team? Buying them books? Sending them to conferences or training courses? Are you hosting brown-bag lunches to discuss technology and development practices?

Software development is risky business. There are many different ways to manage risk, and the agile way favors minimalism as a core approach. Timeboxing, whether in an iteration or in a design meeting, is a very effective way of minimizing risk. Another way of minimizing risk is to choose to write less code, or even better, not to write code. After all, the line of code not written is always correct.

But you already manage risk this way, right? You are constantly thinking about minimizing risk, are not you? You are timeboxing everything, from iterative deliveries and meetings to how long a team member can be stuck on a bug without asking for help?

Similarly, in addition to minimizing risk, you want to maximize business value—the stakeholders need to see a positive return on investment. That means delivering small bits of functionality frequently that help the organization achieve its goals.

Breaking Out of the Rut

Notice that each of these underpinnings of agile demands constant attention and thought. You cannot just run with a set of "standard" practices on autopilot.

You should find yourself thinking. Being skeptical. Re-evaluating. You are doing some set of practices—are they working? Are you getting value for your effort? Is everyone on the team growing and advancing in their own individual careers?

In addition, there are definite warning signs to watch out for, which may indicate you are not working in an agile fashion [3]:

- **Sloganization**—Speech becomes so sloganized that it becomes trivial and eventually loses meaning entirely. Watch for all your favorite agile buzzwords and see if they are being used as meaningless jargon.
- **Confusion between following rules and exercising judgment**—When is it OK to break the rules? All the time? Never? Somewhere in between? How do you know? Some rules probably should never be broken (e.g., check everything into version control, have unit tests, settle on a fixed iteration length) and others may be more flexible (pairing conventions, specifics of customer involvement, etc.).
- **Confusing the model with reality**—A model is not reality. Thinking that an agile project is "supposed to go this way" might be a trap. The only thing it is supposed to do is succeed; everything else is up for grabs.
- **Demanding conformity**—The same standard may not always apply equally in all situations or with all people. You do not want a bunch of monkeys banging on typewriters to churn out code. You want thinking, responsible developers. Do not reward herd behavior or devalue individual creativity.
- **Spelling out too much detail too soon**—Premature optimization is indeed the root of all evil, whether it is in your process or your code. Tying things down too early is not agile, and excessive detail can act like instant glue. Have patience.
- **Oversimplification of complex situations**—"Just follow the process." If it were that easy, anyone could do it. But they cannot. Every project, every situation, is more complex than that. "All you need to do is…" or "Just do this…" are invitations to failure. Do not fall for it.

Become an Agile Practitioner

If any of these warning signs sound familiar, then you and your organization may be stuck in an un-agile rut. Here are some suggestions to help get things back on track.

First, the goal: you (personally) and your teammates (collectively) need to function as a true self-directed, self-correcting, agile machine.

Next, the problem: you are probably operating at an "advanced beginner" level of agile software development. No shame in that, statistically that is where the majority of agile developers will find themselves [3].

To understand this, let us take a look at the Dreyfus Model of Skill Acquisition, and then see how to move to a more advanced level of practice.

The Dreyfus Model

The Dreyfus Model [3] is a helpful way of looking at how people acquire and grow their skill sets. It defines five levels that you experience as you gain abilities at some specific skill.

Here are the levels, with some key features of each:

- **Novice –** Rules-based, just wants to accomplish the immediate goal
- **Advanced beginner** – Needs small, frequent rewards; big picture is confusing
- **Competent** – Can develop conceptual models, can troubleshoot
- **Proficient** – Driven to seek larger conceptual model, can self-correct
- **Expert** – Intuition-based, always seeking better methods

At the advanced beginner stage, you are comfortable enough to sort of muddle your way through. And for most folks, that seems good enough, so any advancement stops there. And unfortunately, this level is not sufficient for self-reflection and correction.

What we need to do is to see how to make the jump from an agile advanced beginner to becoming a competent agile practitioner. If you look at the levels above, one big difference between an advanced beginner and a competent practitioner is the competent's ability to develop and use conceptual models.

So the first thing we need to do is to establish some kind of a mental model about agility.

An Agile Mental Model

Agile is all about taking small bites and focusing on continuous, not episodic, development. You want to do a little of everything all the time, and not face big lumps of major events. That is the "constant" part in this definition, "agile development uses feedback to make constant adjustments in a highly collaborative environment." [4]

The idea of correcting based on continuous feedback is key, but in order to do that successfully you will need to ensure that all the following are true:

1. All of your development and management practices generate continuous, meaningful feedback.
2. You are able to evaluate that feedback appropriately, in context.
3. You are able to change your code, your process, your team, and your interface to the rest of the organization and the customer in order to respond to the feedback.

That is just the mechanical portion. Any problems at this level have to be solved before moving on.

To be agile, you need this kind of base level environment, and then you need to apply an agile mindset. Look at the agile values from the manifesto: [5]

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

If you can proceed in a fashion that honors these values and adheres to that definition, and you are able to respond as needed, then you are headed in a more-or-less agile direction. You have got the basic model of agility in your head, and by asking questions and constantly reviewing what you are doing in terms of this model, you will make progress.

Put it in Practice

That is the theory, anyway. But theory does you no good unless you can put it into your practice. And practice means trying, failing, learning, and trying again. There is no substitute.

Familiar and comfortable ways of doing things may look attractive, but that is a trap–do not fall for it. You will need to venture into the unfamiliar and uncomfortable, and lose a lot of mental baggage while you are at it. Think of the XP maxim and ask yourself, "What is the simplest thing that could possibly work?" What is the least you could possibly get away with? That question works for code, process, documentation, training, QA, and more.

Suppose you need to accomplish a particular goal. Ask yourself how can you accomplish that goal in a lightweight, low-ceremony, agile fashion, and still fully accomplish the goal? As an example, let us look at an aspect of testing that often causes confusion.

You have started a new agile-based project and have been getting everyone up to speed on this new "agile stuff." Now you are tasked with bringing the QA staff into your fledgling environment to do testing. How can you do so in an agile fashion?

Let us clarify what it is they are doing—which is always a fine idea when starting out. We are not talking about "unit testing." Unit testing is mis-named; it is a coding practice, not a testing practice. Yes, developers need to do it, and no, it does not negatively impact your QA staff. So perhaps your development team is doing test-driven or test-first development. That is great, they should be writing unit tests on an ongoing basis, keeping it a continuous practice and avoiding the episodic.

So now you need to bring QA into the fold to do more comprehensive testing, maybe focusing on acceptance testing, performance testing, other non-functional requirements, and so on. Are you going to wait for the end of a major release to give them code to test?

Of course not, that is thinking of the event as a large, discrete episode—that violates the model. You do not want episodic development, you want continuous development. Are you going to wait until the end of an iteration? Perhaps, but that is still waiting and making a bigger event out of the situation than is needed.

Give QA the ability to pick up a testable version right off your continuous build machine. Every build that passes all the automated developer tests is eligible, and QA can grab the latest functioning code base and have a play. They have the opportunity to grab releases every hour, every half-hour, all day long. Or they can wait and grab it once when done (in a non-agile fashion), but the choice is theirs and they can change that choice at will.

Does it Work?

How does this simple example stack up to our mental model?

You have implemented a policy that is lightweight and easy to maintain. You have delivering value continuously, and you are delivering actual working software. QA can make their own decisions about when and what to test. You can experiment with feedback to the development team based on both team's needs. So you are honoring individuals and interactions over some rigid process. You are able to respond to change easily, without affecting the development rhythm, as the two teams are loosely coupled (a great coding technique, works well with organizations as well).

Of course, things rarely go exactly as planned. That is just how the world is. Maybe this scheme does not quite work out for you. That is great! Evaluate the feedback on where the problems are cropping up, and make constant adjustments to try and fix it. That is what agile is supposed to be.

Just as with novice drivers, do not overcorrect, or you could slam into oncoming traffic (metaphorically speaking). Tweak a little, check the feedback, and tweak some more. Even corrections to code or process should not be big, major events. Change a little, all of the time.

And now you are practicing agility, the way we intended.

ABOUT THE AUTHOR

Andy Hunt is a programmer turned consultant, author and publisher. He authored the best-selling book "The Pragmatic Programmer" and six others, was one of the 17 founders of the Agile Alliance, and co-founded the Pragmatic Bookshelf, publishing award-winning and critically acclaimed books for software developers.

The Pragmatic Programmers, LLC
Website: www.PragProg.com
E-mail: andy@pragprog.com
Twitter: @PragmaticAndy
Blog: andy.pragprog.com

REFERENCES

1. Yahoo press release <http://www.ysearchblog.com/2011/05/02/teens-don%E2%80%99t-know-who-osama-bin-laden-is-according-to-yahoo-search-trends Web>.
2. Benner, Patricia E. From Novice to Expert: Excellence and Power in Clinical Nursing Practice. Menlo Park, CA: Addison-Wesley Pub., Nursing Division, 1984. Print.
3. Hunt, Andrew. Pragmatic Thinking and Learning: Refactor Your "wetware". Raleigh, NC: Pragmatic Bookshelf, 2008. Print.
4. Subramaniam, Venkat, and Andrew Hunt. Practices of an Agile Developer: Working in the Real World. Raleigh, NC: Pragmatic Bookshelf, 2006. Print.
5. <agilemanifesto.org>. Web.

Architecting for Large Scale Agile Software Development: A Risk-Driven Approach

Ipek Ozkaya, SEI
Michael Gagliardi, SEI
Robert L. Nord, SEI

Abstract. In this paper, we present lessons we learned while working with a large program, helping it to modernize its very large transaction-based system that operates 24x7, while adopting agile software development. We focus on two agile architecting methods we used that provide rapid feedback on the state of agile team support: architecture-centric risk factors for adoption of agile development at scale and incremental architecture evaluations.

Introduction

Over the past decade of their use, applying agile development methods to large-scale projects has brought its challenges [1, 2]. Challenges are exacerbated when organizations must deal with increased size of software and increased complexity in orchestrating large engineering and development teams, and when they have to ensure that the systems developed will be viable in the market for several decades. Understanding and systematically resolving the challenges becomes even harder for organizations that must adapt their existing processes to agile development (e.g., adapting the DoD acquisition lifecycle or switching from a lengthy development cycle based on other methods to an agile development cycle).

In this paper, we present lessons we learned while working with a large program, helping it to modernize its very large transaction-based system that operates 24x7, while adopting agile software development. We focus on two agile architecting methods we used that provide rapid feedback on the state of agile team support: architecture-centric risk factors for adoption of agile development at scale and incremental architecture evaluations.

Agile project teams recognize that there is a desired software development state that enables them to quickly deliver releases that provide stakeholder value [3]. This involves getting platforms and frameworks, as well as supporting tool environments, practices, processes, and team structures in place to support efficient and sustainable development of features. Conducting a review of architecture-centric risk factors for adoption of agile development at scale uses architecture- centric criteria to examine the organization and project context. The review serves to identify and mitigate key risks to achieving a desired software development state, when there is a need to use agile development and architecture-centric practices in concert. A common adoption risk is losing the focus on architecting activities that help maintain the desired state, enable cost savings, and ensure delivery tempo when other agile adoption activities take center stage [4]. Conducting incremental architecture evaluations that can be incorporated into agile development sprints/iterations assists in mitigating such a risk.

Organizations that must design, develop, deploy, and sustain systems for several decades and manage system and software engineering challenges simultaneously can dispense with neither agility nor attention to enduring design.

What is Scale and Why Manage It?

The amount of software in software-reliant systems has increased tremendously; the software has become more complex, and additional orchestration is needed between the teams that build, integrate, and test the software components. Understanding what is meant by large scale is important, as often several different challenges may be implied that must be teased apart. We investigate scale from three perspectives: scope, team, and time.

Large-scale systems often are large in scope in terms of the amount of new technology being introduced and the number of the features being added, independent components or COTS tools being integrated, users and user types to be accommodated, external systems with which the system communicates, configurations of components that can be configured for deployment, and so on. One or more of these aspects of scope may be present. As scope increases, the team and time dimensions are likely to increase as well.

The team dimension of scale is typically the most often-addressed aspect of scale in agile software development. Practices such as Scrum of Scrums are meant to address orchestration of multiple development teams in concert. There are often teams within or across organizations that are external to the core product development team (e.g., quality assurance, system integration lab, project management, and marketing) that must collaborate and provide input to product development. This brings additional challenges: Careful orchestration is necessary where these teams must seamlessly come up to speed with agile development and collaboration across organizational and lifecycle (e.g., system engineering, assurance) boundaries.

The time dimension of scale relates to both the duration of development and lifecycle time of the system. The system will need to be in development and operation for a longer period than systems to which agile development is typically applied, requiring attention to future changes and possible redesigns, as well as to maintaining several delivered versions. Over time, technology (hardware, software, sensors, effectors, etc.), threats, and features will change in various ways. In response to technology and threat changes, the system will undergo planned or unplanned upgrades. In addition, different planning rhythms may need to be kept in sync, which includes lifecycle budgeting and planning, individual milestone planning, and sprint planning.

Scaling agile projects in any of these dimensions involves a relationship to the architecture of the developed system and the use of architecture practices. For example, when there are multiple teams, the existence of some amount of explicit architecture documentation becomes important to coordinate the work across teams. Or, when a system exists for decades, a focus on the architecture of the system becomes important to ease the evolution of the system over time.

Architecture-centric Risk Factors for Adoption of Agile Development at Scale

When one or more of the scope, team, or time factors are of critical importance, incorporating architecture practices into agile development must take high priority, especially if an organization is new to adopting agile development practices.

In our recent engagement with a large organization, the key dimensions of scale were defined this way: the system has (1) gone through and would go through several technology upgrades, (2) served a large user base, and (3) been in operation and expected to be so for several decades. The organization faced the challenge of moving away from its existing phase-based software development approach to adopting agile development practices, while dealing with several dimensions of scale.

The first step was to conduct a risk analysis that combined a focus on architecture and agile practices. We conducted the risk analysis through several interviews and a scenario walk-through meeting, in addition to examining the working documents of organization-wide adoption plans. We conducted interviews with technical team members as well as managers, where they were in the same sessions as well as in separate sessions. The scenario walk-through meeting presented the team members with possible adoption scenarios and allowed them to analyze their possible outcomes within the organization.

The key areas we investigated included the following: business/acquisition and organizational climate, technology environment, ability to respond to change, project/team support, attention to quality attributes and architecture, customer collaboration, and productivity measures. Here we identify some common risks and factors worthy of attention in each of these areas.

Business/acquisition climate: Without clear identification of business and mission goals that reflect stakeholder concerns and success factors and strategies, adoption of agile practices at scale will entail resolving added challenges. Government organizations must pay special attention to contracting mechanisms. While all mechanisms potentially can be used in software vendor relationships, contracts should not be a barrier to building knowledge within a consistent team and improving communication [5].

Organizational climate: Adoption of agile development practices will require educating the teams about new practices that may not be familiar to a hierarchical organization with phase-based practices and high regulation checkpoints. Prior history of the waterfall processes and arms-length relationships among stakeholders, developers, and acquirers will prolong the time it takes to adopt agile development. These risk factors can affect both decision making in general and technical progress. Such impacts may occur, for example, when organizational barriers make it difficult or impossible to convene committees for technical, architectural, or design reviews.

Technology environment refers to having a robust development and design infrastructure in place to support the development teams. Automated testing and continuous integration practices require ongoing attention to support agile development adequately. To take advantage of agile development practices, such as nightly builds and configuration control, the right infrastructure should be in place and the necessary training provided. Such a technology environment, with automated tests, nightly builds, and configuration control mechanisms, is also tightly related to architectural requirements and the rate of change requests that the system must support. This includes requirements and requests involving new features, but also those that might necessitate architectural changes.

Response to requirements change: The inability to get a handle on the requirements management process in a volatile and large-scale product environment will hinder the success of the project. A special point of caution is to avoid focusing the requirements view on functional requirements; architecturally significant requirements must be continuously addressed as well.

Project/team support highlights attention to granting the authority to the downstream teams and arming them with the necessary skills and knowledge to succeed in an agile context while paying attention to the long-term goals of the system. New roles typically assigned when applying agile methods (such as product owner, Scrum master) have differing responsibilities from the roles in the existing phase-based waterfall program structures. Such differences may necessitate educating not only teams but also management and the customer.

Quality attributes emphasize the architecturally significant requirements of the system. An effort in architecture-focused acquisition can bring forward the key, architecturally significant concerns, such as integration and security. Often at-scale, multiple systems must be orchestrated, which requires continuous management of the key architecturally significant requirements, in addition to development of features.

Architecting activities must be integrated into agile development. The view that keeps these separate often creates silos, architecture conformance issues, and unexpected rework costs in later stages of the development effort. Architectural decisions, however, also must be made with consideration to the goal of iterative development.

Customer collaboration is key to success in any development effort. In an agile development context, customer collaboration is an essential part of the activities, for example, in sprint planning with Scrum. Communication with both internal and external stakeholders must be open and documentation should not be used as a substitute for communication.

Productivity measures in an agile context must incorporate a working system as well as continuous planning. If Scrum is chosen as the project management paradigm for agile development, understanding of relative estimation versus absolute estimation will be necessary. Obtaining the measures requires continuous monitoring and improving of estimates based on lessons learned. Providing working demos to the customer and

establishing done criteria as potentially shippable features must be incorporated into the iteration and release planning cycles. These cycles also require architecting activities to be seamlessly integrated into the sprints.

Figure 1 shows a risk profile example. Having such a risk profile, along with an understanding of the factors contributing to the risks, allows an organization to set priorities in adopting agile practices at scale. This figure also demonstrates a common profile that organizations might face in adopting agile practices at scale. The areas that commonly demonstrate high risks tend to be organizational climate, response to requirements change, and attention to quality attributes and architecture. While business/acquisition context, productivity measures, and project/team support also have issues related to adoption at scale, the risks associated with them would partially resolve if attention were given to the high-risk areas. Customer collaboration and technology environment areas follow as low-risk areas.

Figure 1: Example of risk profile summary

The organization we worked with chose to adopt Scrum as a project management methodology and educate its teams and management in agile development, in addition to keeping a significant focus on architecture.

The typical high-risk nature of architecture and quality attribute areas, as shown in Figure 1, were also issues. In order to ensure that key architectural evolution decisions, risks, and high-priority quality attributes were managed in concert with agile development, we conducted incremental architecture evaluations with the organization, which we describe next.

Incremental Architecture Evaluations

The goal of focusing on architectural evaluation as part of the modernization and agile adoption activities was (1) to identify architectural risks and risk themes of the "as-is" system as well as in migrating the "as-is" system to the target architecture, and (2) to provide actionable recommendations to address the risks themes. The evaluation method was based heavily on the Architecture Tradeoff Analysis Method® (ATAM®) and its principles, but due to constraints, was conducted incrementally [6].

The constraints on the evaluation were twofold. First, the availability of stakeholders and the architects' time to participate in the evaluations was limited to a small number of hours per week. The architects were available for eight hours per week, total. Specific stakeholder's availability was limited to fewer than four hours per week per stakeholder. Second, the documentation for software architecture was inadequate to perform the architecture evaluation per ATAM's criteria. However, we decided to proceed with the evaluation based on the architects' knowledge and to use whatever relevant, useful documentation was available, acknowledging the associated risks of proceeding with the evaluation. This approach enabled us to test an evaluation approach that could also be incorporated seamlessly with other development activities in agile development planning, for example, Scrum sprint and release activities.

Architecture Evaluation Kick-off

It can be challenging to identify the architectural mismatches and impediments to migration from the "as-is" system to the target architecture. This task becomes much more difficult when architecture documentation is lacking.

The evaluation began with a kick-off meeting with the evaluation team, the program office, and the architects. The evaluation team presented the evaluation approach, the program office presented the business drivers, and the architects presented the architecture overview. Despite insufficient architecture documentation, the participants decided to proceed with the evaluation, using the architects' knowledge and any acceptable, relevant documentation that was available. During the architecture evaluation sessions, the scribe would capture any undocumented architecture approaches in the analysis template as the architects described them.

The participants agreed that there would be a series of incremental architecture evaluation sessions on a weekly basis. A notional schedule was developed, starting with mission thread and scenario generation, to be finished within two weeks. Once the mission threads and all of the high-priority scenarios were identified and agreed upon, the schedule for the weekly architecture evaluation sessions would be developed and vetted. The participants also agreed that the evaluation schedule would be flexible and adaptable based on the evaluation progress made on a weekly basis.

Mission Thread and Scenario Generation

The operational end-to-end mission threads were developed by meeting with the operational system manager and technical lead and documenting the most critical threads, using the Mission Thread template for end-to-end mission threads. Table 1 shows a mission thread example, where Table 1(a) shows a summary of the mission thread, including the summary description, and Table 1(b) decomposes the thread into possible steps, with engineering considerations that impact system and software architecture decisions. Table 1(c) specifies the over-arching quality attributes for the mission thread, with engineering considerations that impact the system and software architecture decisions.

Four unique operational mission threads were identified and developed with the operational manager and technical lead and vetted with the system stakeholders. The mission threads established the end-to-end operational context for the evaluation and identified end-to-end quality attribute considerations to be evaluated. This was accomplished in a series of three two-hour meetings with the evaluation team, operational manager, and technical lead in the first week of the evaluation.

Name	Long-term customer order placement with supervisor override
Vignette (Summary Description)	Multi-channel order placement with outsourced fulfillment. The primary business activities are taking orders for products, fulfilling orders, and providing customer service (for example, order changes, return authorization) Orders can be placed over the internet or by calling a customer service agent. Payment processing for credit card payments is handled by a third party (Intuit, ChasePaymentech, etc.). If the payment processing service is not available, the call center agent accepts credit card payments without authorization, and incur the fraud risk. If the service is down for more than 8 hours, the agent accepts credit card information but hold sending the order to the fulfillment service until the payment processing service is restored and payments can be authorized. Order fulfillment is outsourced to a partner, who handles inventory management, order assembly and packing, and shipping and tracking (Amazon, Webgistix, Archway, etc.). There is a service level agreement (SLA) that all orders placed before 4:00pm Eastern Time will be shipped the same day. Our visibility into their process is limited to tracking the shipment on the shipper's web service using the tracking number.
Nodes Actors	Customer Call Center Agent Call Center Supervisor Systems shown in Architecture Overview
Assumptions	1. Customer A is a long-time customer of the company, with high lifetime value. a. What does this mean exactly? More considerations for timeliness, coupons, and so on, need to be analyzed. b. What about high value business customer B? Needs more analysis regarding integration into our system. c. What about family / household value? 2. Shipping charges are computed and applied automatically, but a Supervisor can override this for a particular order. 3. Our visibility into their process is limited to tracking the shipment on the shipper's web service using the tracking number.

Table 1(a): Partially filled mission thread example

Mission Steps	Description	Engineering Considerations, Issues, Challenges
1	Customer A places an order on the web site. The order value qualifies for free shipping.	
2	Payment authorization is obtained for the order. (<5 sec)	
3	The order is recorded and sent for fulfillment.	
4	Customer A's customer history is updated.	
5	….	

Table 1(b): Mission thread steps elaboration

Quality Attribute	Engineering Considerations, Issues, Etc.
Call Center Routing Performance	
Call Center Screen Pop Performance	
Call Center Routing Accuracy	
Order Accuracy	
Time-to-market	
Modifiability	
Testability	
Availability	
Usability for Call Center Agents	
Migratability	
Scalabiility	

Table 1 (c): Quality attribute requirements related to the mission thread

The scenarios were generated in a series of small scenario-generation sessions with the architects and selected stakeholders. The ATAM utility tree was used to capture and elicit scenarios. Figure 2 shows a utility tree example. The utility tree was initially seeded with the quality attributes identified in the business driver's presentation and the operational mission threads. Some scenarios were derived from the mission threads. Each session would begin with a review of the prior session's results. Each session would then elicit quality attribute considerations and scenarios from the attending stakeholders, insert them into the utility tree and then rank each of the scenarios according to degree of difficulty (ranked by architect) and importance (ranked by management)—high, medium or low. The evaluation team, program office, and architects would then discuss the need for any additional scenario generation sessions. Once the utility tree was deemed to be finished, the evaluation team, architects, and program office met to develop an incremental evaluation schedule based on the high-priority scenarios and mission threads. They agreed that the mission threads would be the last evaluation sessions once the high-priority scenarios were all evaluated, in order to evaluate the architecture in an end-to-end manner.

Architecture Evaluation Sessions

There were four mission threads and 59 scenarios generated, based on a utility tree with nine quality attributes. The architecture evaluation sessions covered all of the highly ranked scenarios followed by the four mission threads. The evaluation sessions were grouped based on the quality attributes and the mission threads. The mission thread sessions were held last so that the end-to-end evaluation would follow the scenario-based analyses. The scenario-based evaluation sessions were roughly ordered based on what we considered the more critical quality attributes for the system (e.g., availability, performance, and maintainability).

The session participants decided that there would be three two-hour architecture evaluation sessions per week. The sessions would focus on specific, pre-determined scenarios, based on the quality attribute utility tree and mission threads (e.g., the first week of evaluation sessions would focus on the important availability scenarios, with intent to utilize an evaluation backlog flowing into the next week's session schedule, if necessary).

We expected to analyze two scenarios per session, since the documentation was poor.

Once we finished all the scenario analyses for a quality attribute, we would develop the schedule for the next quality attribute set of scenarios. We repeated this for the entire utility tree and then scheduled the end-to-end mission thread analysis sessions, expecting one thread per two-hour session. The entire set of evaluation sessions required 23 sessions over the course of eight calendar weeks. The evaluation team met to develop risk and non-risk themes after the scenario analyses of each quality attribute and mission thread.

Architecture Evaluation Results

The evaluation sessions resulted in the discovery of numerous, unique, architecture risks and non-risks. The number of risks found was within the typical number and quality of risks found in an architecture evaluation for a system of this type and size. The length of each evaluation session was extended due to the lack of architecture documentation and the need to document roughly the architectural approaches as they arose during the sessions.

The end-to-end mission thread analyses uncovered many risks that were not identified in the scenario-based evaluation sessions. These risks dealt with architectural decisions that supported end-to-end operational processing of data that is difficult to identify when examining parts of a system (as is typical of a scenario walkthrough).

Overall, the incremental architecture evaluation sessions allowed the team to work within the constraints of the engineering team. The schedule developed to conduct such an evaluation approach focused on priority of the scenarios and mission thread steps identified. The approach created tasks that were about two-to-three hours in length and resulted in concrete development and architecture artifacts. This particular evaluation was not incorporated with Scrum sprint activities because the organization had not completed the sprint planning yet. However, the definition of the tasks and the technical outputs created an example of how incremental evaluation could be incorporated into a Scrum development approach by balancing development tasks with a focus on architecture evaluations.

Takeaways

Embracing the principles of agile software development and software architecture provides improved visibility of project status and improved tactics for risk management.

There are different aspects of scale that are manageable with different approaches, such as scope, team, and time.

We see evidence that a systematic architecture-centric review of organizational and project factors, as this organization used, is essential for understanding risks and dealing with the challenges arising in large-scale software development.

We believe the incremental evaluation approach applied to this system could be beneficial in an agile development approach, where small, short architecture evaluation sessions could be implemented in agile sprints. The artifacts we exemplify here, such as mission threads and quality attribute utility trees, could be invaluable in both helping with backlog management and augmenting sprint planning.

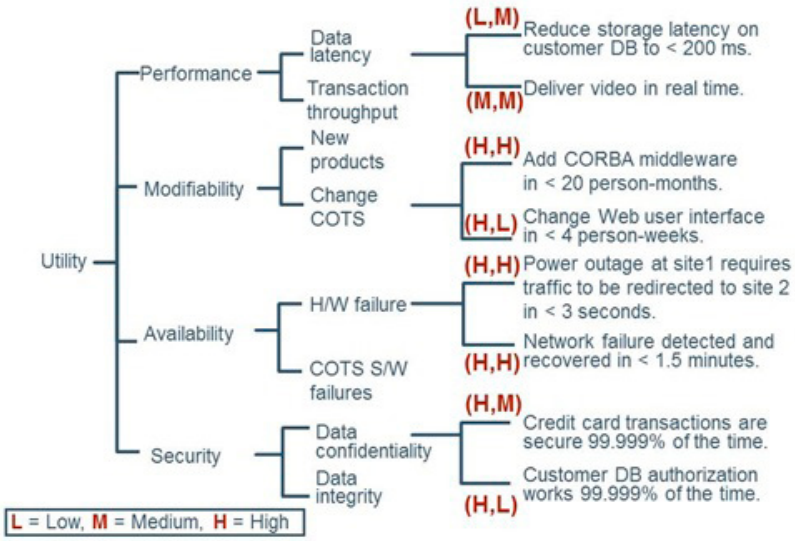


Figure 2: Utility tree example

Acknowledgements/Disclaimers:

Copyright 2013 Carnegie Mellon University

This material is based upon work funded and supported by the DoD 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.

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.

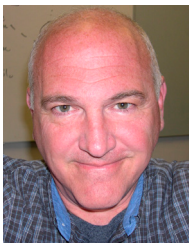
This material has been approved for public release and unlimited distribution. Architecture Tradeoff Analysis Method®, ATAM® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM-0000094

ABOUT THE AUTHORS

Ipek Ozkaya is a senior member of the technical and works to develop empirical methods for improving software development efficiency and system evolution with a focus on software architecture practices, software economics, and requirements management. Her latest publications include multiple articles on these subjects focusing on agile architecting, dependency management, and architectural technical debt. Ozkaya serves on the advisory board of the IEEE Software magazine.

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA
Phone: 412-268-3551
Fax: 412-268-5758
E-mail: ozkaya@sei.cmu.edu



Michael Gagliardi has more than 25 years' experience in real-time, mission-critical software architecture and engineering activities on a variety of DoD systems. He currently works in the SEI Research, Technology, and System Solutions Program on the Architecture-Centric Engineering Initiative, and is leading the development of architecture evaluation and quality attribute specification methods for system and system-of-system architectures.

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

REFERENCES

1. Grant, T. "Navigate the Future of Agile and Lean." Forrester, January 10, 2012.
2. Leffingwell, D. Scaling Software Agility. Upper Saddle River, NJ: Addison-Wesley, 2007.
3. Bachmann, F., Nord, R. L., and Ozkaya, I. "Architectural Tactics to Support Rapid and Agile Stability." Special Issue on Rapid and Agile Stability, CrossTalk, 25.3 (2012): 20-25.
4. Brown, N., Nord, R., and Ozkaya I. "Enabling Agility Through Architecture." CrossTalk 23.6 (2010): 12-17.
5. Lapham, M. A., Miller, S., Adams, L., Brown, N., Hackemack, B., Hammons, C., Levine, L., and Schenker, A. Agile Methods: Selected DoD Management and Acquisition Concerns, October 2011, Technical Note, CMU/SEI-2011-TN-002.
6. Bass, L., Clement, P., and Kazman, R. Software Architecture in Practice, 3rd ed. Addison Wesley, 2012.

Function Points, Use Case Points, Story Points:
Observations From a Case Study

Joe Schofield, International Function Point Users Group
Alan W. Armentrout, Sandia National Laboratories
Regina M. Trujillo, Sandia National Laboratories

Abstract. Software development success, estimation, and measured value continue to challenge projects and organizations today. Schedule and budget seem predictable, perhaps unjustifiably in light of the inability to capture product size expectations early and often. Function Points (FP), Use Case Points (UCP), and Story Points (SP) are three approaches for establishing size (or size of effort). Each of these is applied in a case study to identify similarities and differences. The measure of preference is left for the discerning reader. Limited details about the project, its team size, duration, and goals are intentionally excluded from the research since the intended focus is on comparing the project's measurements.

Introduction

A measure of the size of a software application is required for many different kinds of analysis and decision making. This article looks at three popular methods used today by software project teams to estimate the size and effort required to develop software: SPs, UCPs, and FPs. Of the three, FP measurement is the only one based on an international standard: ISO/IEC 20926:2009. Use case points referred to in this article are based on work described by Clem [1], Cohn [2], and Schneider and Winters [3]. Story points emerge from the Scrum methodology and assign a level of complexity for completion of development tasks [4, 5].

Comparisons among these three measures are rather limited in the literature. Reasons abound as to why the measures should not be compared [6], yet it is natural if not necessary to compare and contrast projects, products, and organizations. The cautions are warranted given historic attempts, for instance, to use backfiring to convert lines of code to function points for comparison. However, because measurements are developed independently and for slightly different purposes, it does not imply that they might not have some discernible and useful association. Until the research has been performed, one cannot dismiss the possibility of meaningful relationships among the measures. During the International Software Measurement and Analysis Conference in Richmond, VA, in September 2011, an "agile working group" identified exploring the relationships among some of these measures as a "priority."

Within this article, these three measures were used to see if insightful comparisons could be made among them. It describes how measures for each of the three methods were calculated for an actual application, and compares them to each other with respect to results, effort to compute, and some, albeit, subjective characteristics. Selecting the one best suited for the current need is left to the discerning reader. A single study does not constitute a statistically valid sample but we are hopeful that it stimulates additional discussion and interest.

Each of the three measures was calculated independently of the other two, and each measure was computed by different individuals. The three measures were determined as follows:

- Story points were created and worked throughout the product development cycle by analysts working with the product owner for the application and were later refined by the developers during their sprint planning sessions. Only implemented story points were included in this study.
- Use case points were developed from use cases produced by a use-case modeler after the product was implemented and were based upon a review of the implemented application. The use cases were validated by the software project team's product verification manager to ensure that the scope was consistent with that used for the story points.
- Function points, like the UCPs, were determined based upon the completed product. A certified, trained, and experienced function point counter developed the measures, once again working with the team's product verification manager to maintain scope consistency with the other two measures.

Note: The traditional lines of code (LOC) are not considered in this article. Jones has called the use of LOC as a measure of software size "professional malpractice." Schofield has demonstrated the statistical unreliability of LOCs. In his worst documented case, the same program, written in the same language, developed by similarly educated developers (most with masters degrees and working as developers), accepted by the same instructor, with counted LOC identically, yielded variations as great as 22:1, and median variations around 9:1 [7].

Applied Story Points – a Closer Look

User story elicitation was the genesis of the SP allocations. It began with input from Subject Matter Experts (SMEs), management and end-users (stakeholders), given to the product owner, where high-level epics were transformed into functional user stories written in the language of business. This activity was followed by weekly grooming sessions where contributors prioritized the list of user stories, decided which are chosen for the next sprint, and then transformed to be sprint-ready, i.e., sprint-sizable stories. These user stories, in basic form, illustrated the functionality that was most significant to the stakeholders. When the bi-weekly Review / Retrospective / Planning (RRP) session commenced, the user stories were brought forth and story points allocated. During the session the development team proposed what it could commit to delivering by the end of the two-week sprint.

This case study's agile software development planning practices included the trendy Fibonacci [8] sequence; this metric is used to estimate the effort of each user story without assigning actual hours. Each number in the Fibonacci series (0, 1, (1), 2, 3, 5, 8, 13, 21, 34, 55 ...) measures relative effort. As an example, an 8 is eight times more effort than a 1. During the weekly grooming sessions, epics (groups of related user stories) were assigned a value using the Fibonacci sequence to provide an overall estimate of effort. Prior to the RRP session, the velocity was defined to suggest the total number of story points the development team could deliver in the sprint. The velocity is calculated based on the team's hours of availability using the following equation:

Projected Sprint Velocity = Average Velocity * (Total Hours this Sprint) / (Max Total Hours)

The resulting Projected Sprint Velocity was used during the RRP session as a basis of planning. Once the lead developer knew the level of commitment, "gut checks" were used by the lead developer and the product owner session to establish the expected velocity for the sprint. After each sprint-ready user story was selected from the repository, arbitrary values were assigned to each user story by each developer to deduce the size (story point count). The resulting value roughly equated to the original "gut check" size as confirmed by other team members through process observation. This behavior is apparently familiar in the agile world, given the following insight: "one characteristic of story points is … that they are a relative measure—which means that they compare the size of one story to another and there is no need for a standard meter value for 1 point. This approach provides flexibility and allows for the gut feeling, experience-based judgments to become statistically accurate." [9]

Note that the words "size" and "effort" are used almost interchangeably by practitioners and Scrum promoters and in fact share overlapping lexical semantics.

Table 1 exhibits the story point distribution over a 12-month development effort of 24 sprints. A total of 50 user stories were produced. Although not reflected in the Table, story point counts remained rather consistent for each two-week sprint.

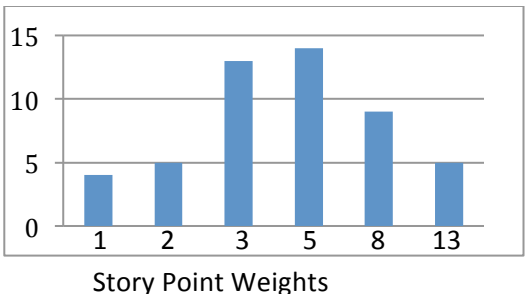


Table 1: Story Point Counts vs. Fibonacci Values

After story points were assigned, each developer decomposed their user story into several tasks, smaller units of measurable work. Following the agile approach [4], each task was assigned a value in terms of hours. These hours reflected the estimated length of time for the developer to complete the task during the two-week sprint. In addition, the features of Business Value and Developer Value were assigned H (high), M (medium), L (low). These two features were only applied during the user story estimation and not used as a means of measure or traceability.

Applied Use Case Points – a Closer Look

The following process was used to develop the use cases and use case points for the case study. The steps used to determine the individual use cases include:

1. Established user goals for the application.
2. Determined individual use cases needed to accomplish the user goals.
3. Outlined the basic flow, subflows, and alternative flows for each use case.

The steps used to calculate the UCPs for the system consisted of summing the unadjusted use case weight (UUCW) and the unadjusted actor weight (UAW) values as detailed by Clem, Cohn, Schneider and Winters, and Zawari [1, 2, 3, 10]. Technical Complexity Factors and Environmental Factors were assumed to evaluate to 1.0, roughly equating to assigning an average impact to each of the factors bearing on the development of the system. This assumption is not consistent with how UCPs are typically calculated. However, it is plausible if UCPs are used to produce a size estimate early in a system's life before the project team is fully assembled and all architectural decisions have been made. This treatment is consistent with excluding the General System Characteristics from the Function Point count below; that is, to assume a mid-value for their sum.

The steps used to determine the UUCW value were as follows:

1. Determined the number of use case transactions for each use case as reflected within its basic flow, subflows, and alternative flows. (See Collaris and Dekker [11] for a discussion on determining use case transactions.)
2. Associated each use case to a use case type using its use case transactions count according to Table 2.
3. Counted the number of use cases by use case type.
4. Multiplied the use case type count by its corresponding weight.
5. Summed the weight-count products across the three use case types to arrive at a UUCW.

Use Case Type	Number of Transactions	Weight
Simple	1 – 3	5
Average	4 – 7	10
Complex	> 7	15

Note: Cohn's, Schneider and Winters' determination of use case types differs from that of Clem's and Zawari's. Cohn's, Schneider and Winters' were used here.

Table 2: Use Case Types

Table 3 shows the UUCW for the case study is 65, as the system has 3 simple use cases, 2 average use cases, and 2 complex use cases.

Use Case Type	Number of Transactions	Weight	Use Case Count	Weight X Count
Simple	1 – 3	5	3	15
Average	4 – 7	10	2	20
Complex	> 7	15	2	30
			UUCW	65

Table 3: Unadjusted Use Case Weight

The steps used to determine the UAW were as follows:

1. Associated each actor to an actor type according to Table 4.
2. Counted the number of actors by actor type.
3. Multiplied the actor type count by its corresponding weight.
4. Summed the weight-count products across the three actor types to arrive at a UAW.

The UAW for the case study is 6, as the system has two actors who interact with the system using a graphical user interface.

The UCP was calculated by adding the UUCW to the UAW. The result of applying the above steps produced an UCP for the case study of 71 (65 + 6).

Applied Function Points – a Closer Look:

The scoping and counting were intended to include all of the functionality delivered to the customer, and only the functionality delivered to the customer. These two considerations are two of Albrecht's three principles of function point counting. The third aspect of Albrecht's [12] seminal characterization was to ensure that function points are developed independent of technology. Table 5 summarizes the data and transactional functions with the subject product.

The values associated with the two data functions and three transactional functions, and their three levels of complexity, as defined in IFPUG's Counting Practices Manual [13] follow.

Multiplying the appropriate values, the case study is measured at 161 unadjusted function points:

(7 * 12) + (5 * 1) + (10 * 3) + (1 * 4) + (2 * 4) + (10 * 3)

The distinction between unadjusted and adjusted is calculated by determining the 14 General System Characteristics (GSCs). Each of these 14 characteristics has a value ascribed to it between 0 and 5 based on a complexity scale that is unique for each characteristic. For the purposes of this example, the GSCs are assumed as the middlemost value, implying no increase or decrease in the UFP count. A similar median value was used for the use case point derivation above. The intention is to keep the counts simple and consistent, and not to introduce the opportunity for variation in assumed characteristics.

"Learning organizations" with historical data have an advantage in being able to review delivery numbers with past performance. Otherwise, organizations undermine their own measurement programs by lessening the value they could be realizing. (As a reminder, the CMMI®-DEV explicitly includes a measurement repository in the Organizational Process Definition process area as specific practice 1.4.) The following organizational values are not intended to represent a particular organization and are used as examples only. Assume for this case study, that the project estimated cost was based on a fixed schedule and whatever work was achievable within its Scrum-based release and sprint planning.

One might ask why size measures are being compared to effort measures. We proffer that because the measures are different it does not follow that useful comparisons cannot be made. Once a size estimate has been calculated one can derive an effort estimate if historical data is available.

Given the existence of related UCP organizational data, similar values could be projected for the cost of a UCP, or expected delivery for 71 UCPs, or defects per UCP, or number of UCPs developed per person month. And of course, similar values could be derived for story points. The presence of that data would likely provide useful performance metrics for monitoring and controlling of the project and for executive decision-making for prioritizing investments in selected development environments and methods.

Actor Type	Description	Weight
Simple	Another system through a defined API.	1
Average	Another system, interacting through a protocol like TCP/IP, or a person interacting through a text-based user interface.	2
Complex	A person interacting through a graphical user interface.	3

Table 4: Actor Types

	Low	Average	High
Internal Logical Files (ILF)	[1]12		
External Interface Files (EIF)	1		
External Inputs (EI)	10	1	
External Outputs (EO)	2		
External Inquiries (EQ)	10		

Table 5: Function Point Data and Functional Transaction Types

	Low	Average	High
ILF	7	10	15
EIF	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

Table 6: Function Point Data and Functional Transaction Values

	Organization's Historical Average	Case Study "Actual"
Function Points (developed) Per Person Month (FPPPM)	25	1.7
Person Months Delivery (for 161 FPs)	13.4	126
Cost Per FP	$425	$14,086
Defect Per FP	0.13	.92

Total Function Points: 161; Total Person Months: 126; Total Cost $2,268,000; Total Defects: 148 (Defects from peer reviews and test—pre-release)

Table 7: (Example) Organizational Historic Values Compared to Project and Actual Project Values

Conclusions

Given the data in Table 7, it is hard to imagine that a project that deviates this much from past performance would be allowed to continue "as is." So while the data in the case study may seem extreme, a better question might be, "at what point in a project's life does one initiate corrective action?" Certainly variation outside a range should trigger corrective action; failure to adhere to that range or percent can be a sign of "inattentiveness." Still other implications related to the team and personnel performance begin to emerge; but, that is not the target of this case study.

We are hopeful that the following characterizations advance the understanding and serious analysis of measurement methods. We have at least documented one such case.

Characteristic	Function Points	Use Case Points	Story Points
Useful at the project level for estimating or planning	With historical FP data	With historical UCP data	With historical SP data
ISO / Standards based	ISO 20926	no	no
Captures customer view	Expected	Expected	Definitely
Useful for benchmarking outside the company	If used as intended	Could be	Not intended to be used across projects
Easy to calculate	Less so	More so	Yes
Easy to validate for repeatability / consistency	More so	More so	Less so
Objectivity	More so	More so	Less so (team / team member variability)
Technologically independent	Yes	Yes	Maybe
Functional measurement to customer	Yes	Yes	Not exclusively (may include refactoring, design, and other work)
Time to derive estimate	Less than one hour	Less than 30 minutes	Indeterminable but significantly larger than other measures

Table 8: Comparing Function Points, Use Case Points, and Story Points

While there are no direct correlations of function points to use case points, or vice versa, to our understanding, effort estimates may reveal useful insights. Therefore, the following effort calculations were derived for the case study:

- 161 function points / 25 function points per month (a historical organizational average) / 12 months per year = .54 year of effort
- 71 use case points * 20 hours per use case point (the low value suggested by Schneider and Winters [3]) / 2080 work hours per year (52 weeks * 40 hours per week) = .68 year of effort
- 71 use case points * 28 hours per use case point (the high value suggested by Schneider and Winters [3]) / 2080 work hours per year = .96 year of effort

Note: The effort value for Story Points is not rendered because initial values during the development of the product backlog, subsequently re-calculated by the development team during sprint planning, are subject to additional variables like refactoring, grooming, and the developer's estimation capability.

Productivity measurement is seldom a topic welcomed with open arms by project teams. Early reticence to measurement for organizations and teams is often characterized by comments like:

- Measurement is hard; (though it is even harder if postponed or ignored).
- We make only limited decisions based on measurement data, thus the need to collect it seems specious.
- How do my numbers compare to others? We may need to adjust them.
- Our numbers could be used against us.
- Our numbers could be better; for now, they are good ballpark estimates.

A Time Magazine article "Good Guess—Why we should not underestimate the alue of estimating" examines the importance of estimating. The article encourages parents to incorporate estimating into their children's thinking at a very early age citing among others, a study from Carnegie Mellon [14]. Contrast this thinking with attempts in the agile world to shift the discussion towards "level of difficulty" versus a delivery time estimate [5].

Two of the purposes of "counting" in the software development world are to provide insight for the awaiting customer and improvement across various development activities. Use Case Points, Story Points, and Function Points are three techniques that can provide measurement insight for software projects. It is less obvious that each of these provide similar value to the customer or organization for scheduling. Perhaps the "maturity" of the organization and the culture defines the "tolerance level" (adoptability?) of organizational measures. An absence of relevant comparisons has been published thus far; rendering the verification of relevant measures difficult. It is too early to suggest that valid comparisons or the emergence of preferences among these measurements are unlikely. Comparisons are more easily facilitated by industry standards as is the case for Function Points.

Acknowledgements/Disclaimers:

The authors wish to acknowledge Don Likfe for his review of the representation of the Fibonacci data in the paper.

The "Applied Story Points – A Closer Look" and "Applied Use Case Points – a Closer Look" sections of the article were funded by Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

ABOUT THE AUTHORS

Joe Schofield is the President of the International Function Point Users Group. He retired from Sandia National Laboratories as a Distinguished Member of the Technical Staff after a 31-year career. During twelve of those years he served as the SEPG Chair for an organization of about 400 personnel which was awarded a SW-CMM Level 3; he continued as the migration lead to CMMI Level 4 until his departure. Joe has taught graduate courses since 1990. He has over four dozen published papers, conference presentations and keynotes—including contributions to four IT measurement books. He is a SEI-authorized Instructor for the Introduction to the CMMI and two other SEI courses, CSQA, CFPS, CSMS, and a LSS BB. Joe is a frequent presenter in the Software Best Practices Webinar Series sponsored by Computer Aid, Inc. Joe completed his Master's degree in MIS at the University of Arizona in 1980.

Alan W. Armentrout, CPA, Inactive, is a Distinguished Member of the Technical Staff at Sandia National Laboratories. He has worked at Sandia for more than 23 years analyzing, designing, and developing information systems. Alan is a Certified Professional for Requirements Engineering, Foundation Level with the International Requirements Engineering Board. He completed his Master's Degree in Management Information Systems at Colorado State University in 1988. Prior to completing that degree, Alan worked six years as an accountant with an international petroleum company.

Regina M. Trujillo is a Member of Technical Staff at Sandia National Laboratories where she specializes in verification and validation of software systems, from complex satellite groundstations to simple web applications. She also has knowledge and experience in running security, load, performance, and transactional analysis tests against these software systems to ensure compliance with corporate standards. Regina has a Bachelor's degree in Computer Information Systems from New Mexico Highlands University.

REFERENCES

1. Roy Clem; "Project Estimation with Use Case Points"; 03/22/2005; retrieved from <http://www.codeproject.com/KB/architecture/usecasep.aspx> 12/2011
2. Mike Cohn; Estimating with Use Case Points; <www.methodsandtools.com>, retrieved 8/29/2011
3. Geri Schneider and Jason P. Winters; Applying Use Cases, Second Edition: A Practical Guide; Addison-Wesley; 2001.
4. What is a Story Point?; November 13, 2007; retrieved from <http://agilefaq.wordpress.com> 12/23/2011
5. Mike Cohn; "It's Effort not Complexity"; 9/19/2011; retrieved from <http://blog.mountaingoatsoftware.com/tag/story-points> 1/16/2012
6. IFPUG Bulletin Board; "Converting Use Case Points to IFPUG Function Points"; posted September 2006; retrieved 12/2011
7. Joe Schofield; "The Statistically Unreliable Nature of Lines of Code"; CrossTalk; April, 2005
8. <http://en.wikipedia.org/wiki/Fibonacci_number>; retrieved 1/24/2012
9. Jack Milunsky; The significance of Story points; March 20, 2009; retrieved from <http://agilesoftwaredevelopment.com/blog/jackmilunsky/significance-story-points> 12/2011;
10. Abdollah Zawari; "Project Estimation with Use Case Points using Enterprise Architect," retrieved from <http://community.sparxsystems.com/tutorials/project-management/project-estimation-use-case-points-using-enterprise-architect-ea> 12/2011
11. Remi-Armand Collaris and Eef Dekker; "Software Cost Estimation Using Use Case Points: Getting Use Case Transactions Straight", 03/13/2009; retrieved from <http://www.ibm.com/developerworks/rational/library/edge/09/mar09/collaris_dekker/> 12/2011
12. A. J. Albrecht, "Measuring Application Development Productivity," Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, California, October 14–17, IBM Corporation (1979), pp. 83–92.
13. International Function Point Users Group (IFPUG); Counting Practices Manual, v 4.3; January, 2010
14. Annie Murphy Paul; "Good Guess – why we shouldn't underestimate the value of estimating"; TIME; (December 12, 2011): 68

NOTES

1. This includes treating the source of many dropdowns as customer-requested ILFs since the customer will maintain this data in a future release; thus the apparent disparity among ILFs and EIs.

International Systems and Software Engineering Standards for Very Small Entities

Claude Y. Laporte, École de technologie supérieure
Rory V. O'Connor, Dublin City University
Gauthier Fanmuy, ADN

Abstract. Very Small Entities (VSEs) developing systems or software are very important to the military since the components they develop are often integrated into products made by larger organizations. Failure to deliver a quality product on time and within budget may threaten both customers and suppliers. One way to mitigate these risks is to put in place proven engineering practices. ISO has approved recently the publication of standards and technical reports, known as ISO/IEC 29110, to address the needs of VSEs.

Introduction

More than ever, integrators of military systems depend on their numerous suppliers to deliver sub-systems meeting evolving requirements correctly, predictably, rapidly, and cost effectively. A supply chain of a large system often has a pyramidal structure. If an undetected defect is left in a low level component, once this component is integrated into a higher level component, the defect may still be undetected. For example, as illustrated in Figure 1, a large manufacturer integrated into one of its products a component, with an undetected software error, which was produced by one of its lowest-level suppliers. This defective component resulted in a loss of millions of dollars by the manufacturer.

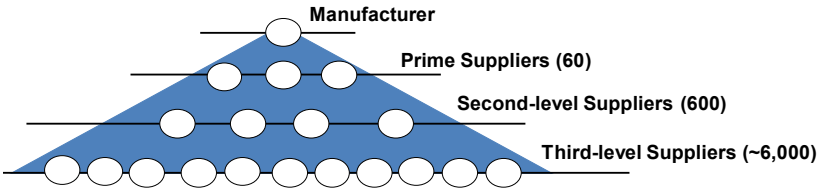


Figure 1: Example of the supply chain of a large manufacturer (adapted from [1])

The ability of organizations to compete, adapt, and survive depends increasingly on quality, productivity, cycle time and cost. Systems and software are getting bigger and more complex every year. As an example, a top of the line cars have up to 100 million lines of code, 80 processors and 5 bus systems [2].

Industry recognizes the value of VSEs, i.e. enterprises, organizations, projects or departments with up to 25 people [3], in contributing valuable products and services. There is a need to help these organizations understand the benefit of the concepts, processes and practices described in systems and software engineering standards, and to help them in their implementation. At every level of the supply chain, illustrated in figure 1, we find VSEs since a system integrator as well as its prime suppliers have also very small projects.

Research shows that small and very small enterprises can find it difficult to relate to ISO standards to their business needs and to justify the application of the standards to their business practices [4]. Most of these enterprises do not have the expertise or can not afford the resources—in number of employees, cost, and time—or see a net benefit in establishing lifecycle processes. There is sometimes a disconnect between the short-term vision of an enterprise, looking at what will keep it in business for another six months or so, and the long-term or mid-term benefits of gradually improving the ways the enterprise can manage its development and maintenance processes. A primary reason cited by many small enterprises for this lack of adoption of systems or software engineering standards, is the perception that they have been developed by and for large companies and not with very small organizations in mind [5]. To date, VSEs have no or very limited ways to be recognized, by large organizations, as enterprises that produce quality products within budget and calendar in their domain and may therefore be cut off from some economic activities. Accordingly there was a need to help VSEs understand and use the concepts, processes and practices proposed in the ISO/IEC JTC1/SC7's[1] international engineering standards.

The recently published set of ISO/IEC 29110 international standards (IS) and technical reports (TR) are 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)[2], are targeted at VSEs which do not have experience or expertise in selecting, for a specific project, the appropriate processes from lifecycle standards such as ISO/IEC 12207 [6] or ISO/IEC 15288 [7], tailor them to the needs of a specific project.

In the next section, a high level summary of the approach used to develop the ISO/IEC 29110 standard and discuss some of its key concepts, including project management and software implementation processes. We will then present the initial support work on deployment assistance for VSE in using this standard and finish by discussing the planned future work.

The WG24 Approach to the Development of Standards for VSEs Developing Software

Since an international standard dedicated to the software lifecycle processes was already available, i.e. ISO/IEC 12207, WG24 used the concept of ISO standardized profiles (SP) to develop the new standards for VSEs developing software. From a practical point of view, a profile is a kind of matrix which identifies precisely the elements that are taken from existing standards from those that are not. The overall approach followed by WG24 to develop this new standard for VSE consisted of the following steps:

1. Develop a set of profiles for VSEs not involved in critical software development,
2. Select the ISO/IEC 12207 process subsets applicable to VSEs having up to 25 people,
3. Select the description of the products, to be produced by a project, using ISO/IEC 15289 standard [8],
4. Develop guidelines, checklists, templates, and examples to support the subsets selected.

Generic Profile Group

Profile Groups are a collection of profiles. The Generic Profile Group has been defined as applicable to a vast majority of VSEs that do not develop critical systems or critical software [9]. This Profile Group is a collection of four profiles (Entry, Basic, Intermediate, Advanced) providing a progressive approach to satisfying a vast majority of VSEs. VSEs targeted by the Entry Profile are VSEs working on small projects (e.g. at most six person-months effort) and for start-up VSEs. The Basic Profile describes software development practices of a single application by a single project team of a VSE. The Intermediate Profile is targeted at VSEs developing multiple projects within the organizational context taking advantage of it. The Advanced Profile is targeted to VSEs which want to sustain and grow as an independent competitive software development business. Table 1 illustrates this profile group as a collection of four profiles, providing a progressive approach to satisfying the requirements of a profile group.

Generic Profile Group			
Entry	Basic	Intermediate	Advanced

Table 1: Graduated profiles of the Generic profile group (adapted from [9])

The ISO/IEC 29110 standards and technical reports targeted by audience are described in Table 2.

The set of documents, listed in table 2, for the Basic profile ([9], [10], [11], [12], [13]) were published in 2011. At the request of WG24, all ISO/IEC 29110 TRs are available at no cost from ISO <http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html>. The Management and Engineering Guide, the most valuable document for VSEs, has being translated to French by Canada and to Spanish by Peru and adopted as a Peruvian national standard. The set of 5 documents has been translated to Portuguese by Brazil and adopted as a Brazilian national standard. Japan has translated and adopted ISO/IEC 29110 as a Japanese national standard. The Management and Engineering guide of the Entry profile has been published in September 2012 in English [14] and in French [15].

Overview of the Basic Profile for VSEs Developing Software

The purpose of the Basic Profile is to define Software Implementation (SI) and Project Management (PM) processes from a subset of ISO/IEC 12207 and ISO/IEC 15289 appropriate for VSEs. The main reason to include 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 relationship between the SI and the PM processes is illustrated in Figure 2.

ISO/IEC 29110	Title	Target audience
Part 1	Overview	VSEs, customers, assessors, standards producers, tool vendors, and methodology vendors.
Part 2	Framework and taxonomy	Standards producers, tool vendors and methodology vendors. Not intended for VSEs.
Part 3	Assessment guide	Assessors, customers and VSEs
Part 4	Profile specifications	Standards producers, tool vendors and methodology vendors. Not intended for VSEs.
Part 5	Management and engineering guide	VSEs and customers

Table 2: ISO/IEC 29110 target audience (adapted from [3])

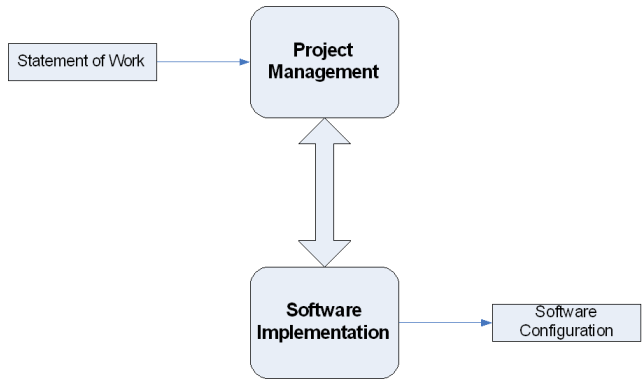


Figure 2: Basic profile process relationship [12]

As illustrated in figure 2, the customer's statement of work is used to initiate the PM process. The project plan will be used to guide the execution of the software requirements analysis, software architectural and detailed design, software construction, and software integration and test, and product delivery activities. Verification, validation, and test tasks are included in the SI process. The PM process closure activity will deliver the Software Configuration (i.e. a set of software products) and will obtain the customer's acceptance to formalize the end of the project.

Overview of the Project Management Process

The purpose of the Project Management process is to establish and carry out the tasks of the software implementation project in a systematic way, which allows compliance with the project's objectives in terms of expected quality, time, and costs [12]. The seven objectives of the PM process are listed in table 3.

PM.O1. The Project Plan for the execution of the project is developed according to the Statement of Work and reviewed and accepted by the Customer. The tasks and resources necessary to complete the work are sized and estimated.
PM.O2. Progress of the project is monitored against the Project Plan and recorded in the Progress Status Record.
PM.O3. The Change Requests are addressed through their reception and analysis. Changes to software requirements are evaluated for cost, schedule and technical impact.
PM.O4. Review meetings with the Work Team and the Customer are held. Agreements are registered and tracked.
PM.O5. Risks are identified as they develop and during the conduct of the project.
PM.O6. A software Version Control Strategy is developed. Items of Software Configuration are identified, defined and baselined. Modifications and releases of the items are controlled and made available to the Customer and Work Team including the storage, handling and delivery of the items.
PM.O7. Software Quality Assurance is performed to provide assurance that work products and processes comply with the Project Plan and Requirements Specification.

Table 3: Objectives of the Project Management process of the Basic Profile [12]

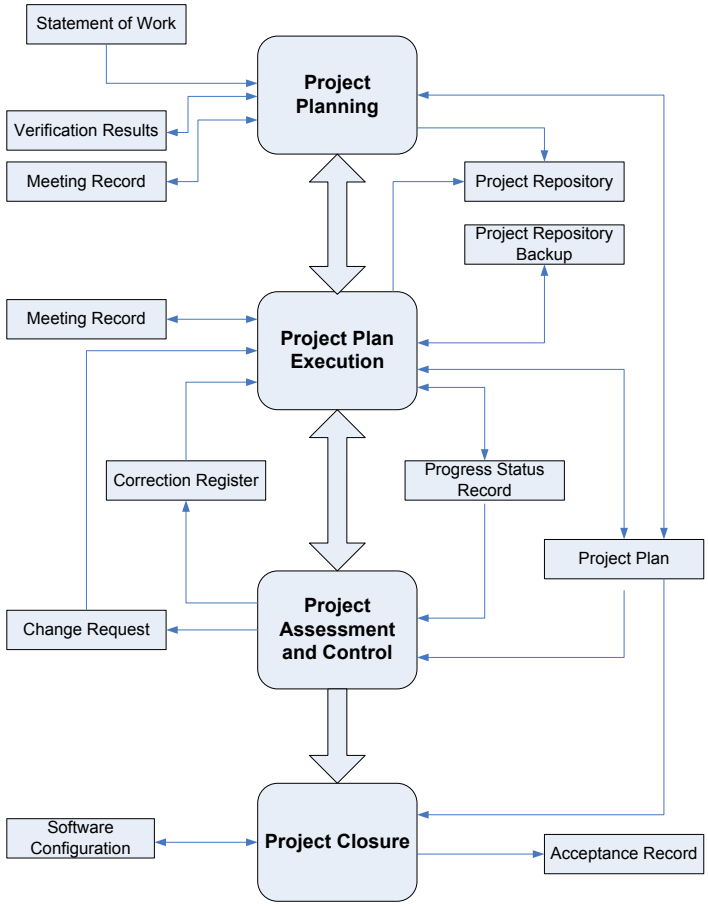


Figure 3: Project management process diagram for software [12]

Role	Task List	Input Products	Output Products
PM CUS	PM.1.2 Define with the Customer the Delivery Instructions of each one of the Deliverables specified in the Statement of Work.	Statement of Work [reviewed]	Project Plan - Delivery Instructions
PM CUS	PM.1.14 Review and accept the Project Plan. Customer reviews and accepts the Project Plan, making sure that the Project Plan elements match with the Statement of Work.	Project Plan [verified]	Meeting Record Project Plan [accepted]

Table 4: Example of 2 tasks of the Project Planning Activity [12]

SI.O1. Tasks of the activities are performed through the accomplishment of the current Project Plan.
SI.O2. Software requirements are defined, analyzed for correctness and testability, approved by the Customer, baselined and communicated.
SI.O3. Software architectural and detailed design is developed and baselined. It describes the Software Components and internal and external interfaces of them. Consistency and traceability to software requirements are established.
SI.O4. Software Components defined by the design are produced. Unit test are defined and performed to verify the consistency with requirements and the design. Traceability to the requirements and design are established.
SI.O5. Software is produced performing integration of Software Components and verified using Test Cases and Test Procedures. Results are recorded at the Test Report. Defects are corrected and consistency and traceability to Software Design are established.
SI.O6. A Software Configuration, that meets the Requirements Specification as agreed to with the Customer, which includes user, operation and maintenance documentations, is integrated, baselined and stored at the Project Repository. Needs for changes to the Software Configuration are detected and related change requests are initiated.
SI.O7. Verification and Validation Tasks of all required work products are performed using the defined criteria to achieve consistency among output and input products in each activity. Defects are identified, and corrected; records are stored in the Verification/Validation Results.

Table 5: Objectives of the Software Implementation process of the Basic Profile [12]

Figure 3 illustrates the 4 activities of the project management process as well as their input and output products. Each activity is composed of tasks. The task description does not impose any technique or method to perform it.

For illustration purposes, two tasks of the Project Planning activity are listed in Table 4. On the left side of the table are listed the roles involved in a task. The project manager (PM) and the customer are involved in these 2 tasks.

The customer is involved, during the execution of the project, when he submits change requests, during project review meetings, for the validation and approval of the requirements specifications and for the acceptance of the deliverables.

Overview of the Software Implementation Process

The purpose of the (SI) process is to achieve systematic performance of the analysis, design, construction, integration, and test activities for new or modified software products according to the specified requirements [12]. The seven objectives of the SI process are listed in table 5.

Figure 4 illustrates the 4 activities of the SI process as well as their input and output products. Even though, a sequential view is presented in figures 3 and 4, ISO/IEC 29110 is not intended to preclude the use of different lifecycles such as waterfall, iterative, incremental, evolutionary or agile.

Deployment Packages to Facilitate the Implementation of ISO/IEC 29110

As a novel approach taken to assist VSEs with the deployment of ISO/IEC 29110 and to provide guidance on the actual implementation of the Management and Engineering Guides in VSEs, a series of Deployment Packages (DPs) have been developed to define guidelines and explain in more detail the processes defined in the ISO/IEC 29110 profiles [16]. The elements of a typical DP are: description of processes, activities, tasks, steps, roles, products, templates, checklists, examples, references and mapping to standards and models, and a list of tools. The mapping shows that a deployment package has explicit links to standards, such as ISO/IEC 12207, or models, such as the CMMI® for Development [17]. Hence by implementing a DP, a VSE can see its concrete step to achieve or demonstrate coverage [18].

DPs were designed such that a VSE can implement its content, without having to

implement the complete framework, i.e. of the management and engineering guide, at the same time. A set of nine DPs have been developed to date and are freely available from [19]. Figure 5 illustrates the set of DPs developed to support the Basic Profile.

The Development of Systems Engineering Standards and Guides for VSEs Developing Systems

In 2009, the International Council on Systems Engineering (INCOSE) Very Small and Micro Entities Working Group (VSME) was established to evaluate the possibility of developing a standard, using the Generic profile group scheme of the ISO/IEC 29110 series, based on ISO/IEC 15288, for organizations developing systems. The ISO/IEC 29110 standard is targeted at VSEs which do not have experience or expertise in tailoring ISO/IEC 15288 [20].

In November 2011, WG24 met in Ireland to launch the official development of the systems engineering ISs and TRs for VSEs. Delegates from Brazil, Canada, France, Japan, Thailand, United States and INCOSE participated to the first meeting. A first draft was sent for a round of review within ISO in January 2012. More than 450 comments have been submitted by seven countries. A second draft [21] was sent for a second round of review in December 2012. Less than 150 comments have been submitted and will be processed at the next WG meeting. A third and final cycle of review should start in July. The systems engineering Basic profile should be published by ISO either late 2013 or early 2014. A set of systems engineering DPs is also under development by systems engineers members of INCOSE to support the Basic Profile.

Conclusion and Future Work

A large majority of organizations worldwide have up to 25 people. The collection of ISO/IEC JTC1 SC7 standards was not easily applied in VSEs, which generally find engineering and management standards difficult to understand and implement. WG24 developed a series of ISO/IEC 29110 ISs and TRs for VSEs involved in the development of systems or software.

As ISO/IEC 29110 is an emerging standard there is much work yet to be completed. The main remaining work item is to finalize the development of the remaining two software

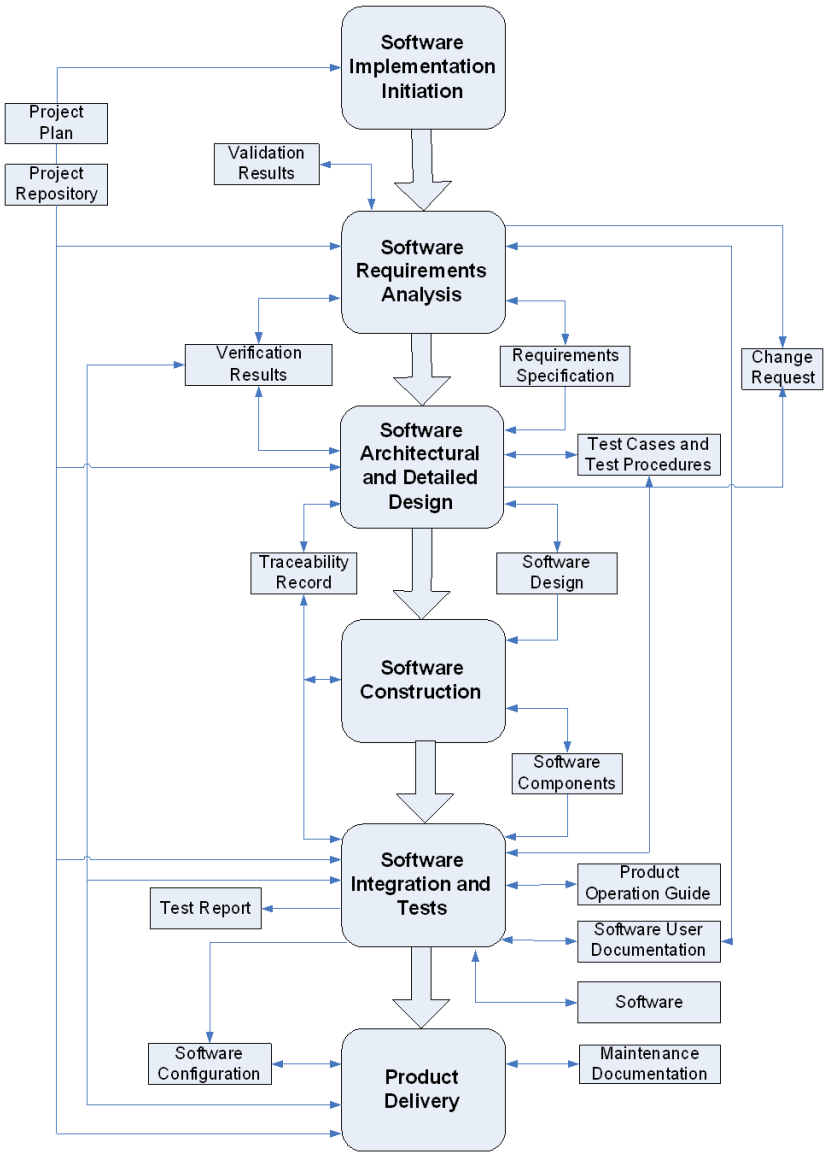


Figure 4: Software implementation process diagram [12]

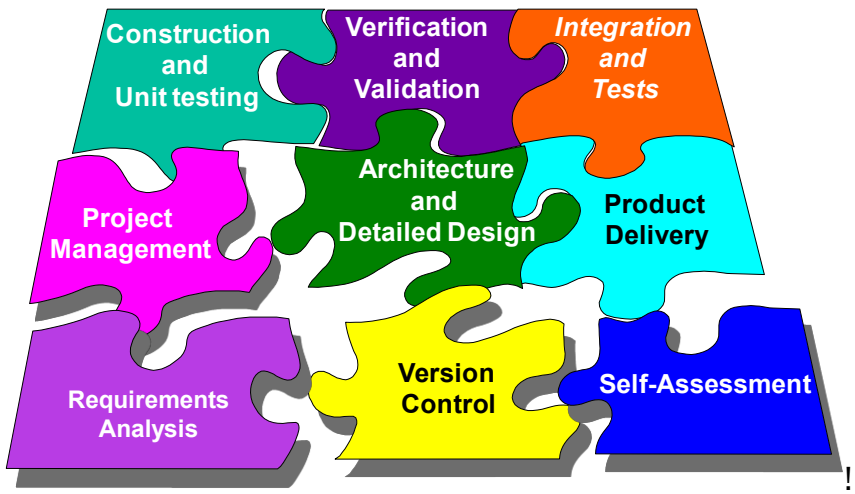


Figure 5. Deployment Packages to support the Software Basic Profile [18]

The publication by ISO, of the systems engineering Basic profile, is anticipated for either late 2013 or early 2014. Similar to the existing set of software ISO/IEC 29110 TRs, the Management and Engineering Guide for systems engineering should also be made available at no cost by ISO.

profiles of the Generic Profile Group: (a) Intermediate—management of more than one project and (b) Advanced—business management and portfolio management practices. Once these software profiles are ready, WG24 will develop matching systems engineering profiles for VSEs.

For most enterprises, but in particular for VSEs, international certifications can enhance credibility, competitiveness and access to national and international markets. Brazil has led the development of an ISO/IEC 29110 certification process. An ISO/IEC 29110 auditor should be competent in auditing techniques, have expertise in ISO/IEC 29110 and have experience in software development. For VSEs, such a certification should not be too expensive and short. The certification process has been successfully piloted in a few VSEs. For these pilots, it took about 4 staff-days of work by the auditors.

The publication by ISO, of the systems engineering Basic profile, is anticipated for either late 2013 or early 2014. Similar to the existing set of software ISO/IEC 29110 TRs, the Management and Engineering Guide for systems engineering should also be made available at no cost by ISO. A set of DPs, to support the future systems engineering standard, will be made freely available to VSEs on public Web sites. These DPs will use as a main reference the INCOSE Handbook [22].

Since many VSEs developing systems are also involved in the development of critical systems, WG24 will conduct an analysis to determine if a set of systems/software standards for VSEs developing critical systems should be developed.

Additional Information

The following Web site provides more information about ISO/IEC 29110:

<http://profs.logti.etsmtl.ca/claporte/English/VSE/index.html>

Disclaimer:

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

ABOUT THE AUTHORS

Dr. Claude Y Laporte has worked in defense and transportation organizations for more than 20 years. He is a professor, since 2000, at the École de technologie supérieure, a 6,000-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 textbook about software quality assurance which should be published by John Wiley & Sons in 2013. Website address: <http://profs.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
E-mail: Claude.Y.Laporte@etsmtl.ca

Dr. Rory V. O'Connor is a Senior Lecturer in Software Engineering at Dublin City University (Ireland) and a Senior Researcher with Lero, The Irish Software Engineering Research Centre. In addition he is Ireland's Head of delegation to ISO/IEC JCT1/SC7. His research interests are centered on the processes 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
E-mail: roconnor@computing.dcu.ie

Gauthier Fanmuy is a Department Director at ADN <http://www.adneurope.com>, a Systems Engineering consulting company. He has worked in the Automotive Industry at PSA Peugeot Citroen as a System Engineering Expert, and in the Aeronautic Industry at Dassault Aviation as system engineer and as a project manager. He is Deputy Technical Director of AFIS (French Association on Systems Engineering, <http:www.afis.fr> and AFIS representative at AFNOR. He is Associate Technical Director for Industry in INCOSE.

ADN
Systems Engineering Department Director
17 rue Louise Michel
92300 Levallois Perret - France
E-mail: gauthier.Fanmuy@adn.fr

REFERENCES

1. Shintani, K., Empowered Engineers are Key Players in Process Improvement, Presentation at the First International Research Workshop for Process Improvement in Small Settings, Software Engineering Institute, CMU/SEI-2006-SR-01, Pittsburgh, PA, 2006.
2. Charette, R. N., Why Software Fails, IEEE Computer Society, Spectrum, September 2005, pp 42-49.
3. Laporte, C.Y., Alexandre, S., and O'Connor, R. A Software Engineering Lifecycle Standard for Very Small Enterprises, R.V. O'Connor et al (Eds) Proceedings of EuroSPI Springer-Verlag, CCIS Vol. 16, pp. 129-141, 2008.
4. Coleman, G., O'Connor, R., Software Process in Practice: A Grounded Theory of the Irish Software Industry. In: Richardson, I., Runeson, P., Messnarz, R. (eds.) EuroSPI 2006. LNCS, vol. 4257, pp. 28–39. Springer, Heidelberg (2006)
5. O'Connor, R. and Laporte, C.Y., Deploying Lifecycle profiles for Very Small Entities: An Early Stage Industry View, Proceedings of 11th International SPICE Conference on Process Improvement and Capability dEtermination, CCIS Vol. 155, Springer-Verlag, May 2011.
6. ISO/IEC 12207:2008, Information technology – Software lifecycle processes. International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland.
7. ISO/IEC 15288:2008, Systems and software engineering - System lifecycle processes, International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland, 2008.
8. ISO/IEC 15289:2011, Systems and software engineering – Content of lifecycle information products (documentation), International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland, 2011.
9. ISO/IEC 29110-2:2011 Software Engineering - Lifecycle Profiles for Very Small Entities (VSEs) - Part 2: Framework and Taxonomy, Geneva: International Organization for Standardization (ISO), 2011. Available from ISO at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51151
10. ISO/IEC TR 29110-1:2011, "Software Engineering - Lifecycle Profiles for Very Small Entities (VSEs) - Part 1: Overview". Geneva: International Organization for Standardization (ISO), 2011. Available at no cost from ISO at: http://standards.iso.org/ittf/PubliclyAvailableStandards/c051150_ISO_IEC_TR_29110-1_2011.zip
11. ISO/IEC TR 29110-3:2011, "Software Engineering - Lifecycle Profiles for Very Small Entities (VSEs) - Part 3: Assessment Guide". Geneva: International Organization for Standardization (ISO), 2011. Available at no cost at: http://standards.iso.org/ittf/PubliclyAvailableStandards/c051152_ISO_IEC_TR_29110-3_2011.zip
12. ISO/IEC TR 29110-5-1-2:2011- Software Engineering - Lifecycle Profiles for Very Small Entities (VSEs) - Part 5-1-2: Management and engineering guide - Generic profile group: Basic profile, International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland. Available at no cost from ISO at: http://standards.iso.org/ittf/PubliclyAvailableStandards/c051153_ISO_IEC_TR_29110-5-1_2011.zip
13. ISO/IEC 29110-4-1:2011, "Software Engineering -- Lifecycle Profiles for Very Small Entities (VSEs) - Part 4-1: Profile specifications: Generic profile group". Geneva: International Organization for Standardization (ISO), 2011. Available from ISO at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51154
14. ISO/IEC TR 29110-5-1-1:2012- Software Engineering - Lifecycle Profiles for Very Small Entities (VSEs) - Part 5-1-1: Management and engineering guide - Generic profile group: Entry profile, International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland. Available at no cost from ISO at: http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html
15. ISO/IEC TR 29110-5-1-1:2012, Ingénierie du logiciel – Profils de cycle de vie pour les très petits organismes (TPO) –Partie 5-1-1: Guide de gestion et d'ingénierie– Groupe de profils génériques : Profil d'entrée. International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland. Available at no cost from ISO at: http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html
16. Laporte, C.Y., Alexandre, S., Renault, A., The Application of International Software Engineering Standards in Very Small Enterprises, Software Quality Professional Journal, ASQ, Vol. 10, No 3, June 2008, p 4-11.
17. Software Engineering Institute. (2010). CMMI for Development, Version 1.3, Pittsburgh PA: Carnegie Mellon University. CMU/SEI-2010-TR-033.
18. Laporte, C.Y., Palza Vargas, E., The Development of International Standards to facilitate Process Improvements for Very Small Enterprises, Book: "Software Process Improvement and Management: Approaches and Tools for Practical Development" IGI Global Publisher. USA. 2012, p 34-61.
19. ISO/IEC JCT1/SC7 Working Group 24 Deployment Packages repository, [online] available from http://profs.logti.etsmtl.ca/claporte/English/VSE/index.html
20. Laporte, C. Y., Fanmuy, G., Ptack, K., The Development of Systems Engineering International Standards and Support Tools for Very Small Enterprises, International Council on Systems Engineering (INCOSE) International Symposium, July 9 – 12, 2012, Rome Italy.
21. ISO/IEC DTR 29110-5-6-2: 2012 - Systems Engineering – Lifecycle Profiles for Very Small Entities (VSEs) - Management and engineering guide: Generic profile group: Basic profile, International Organization for Standardization/International Electrotechnical Commission: Geneva, Switzerland.
22. Systems Engineering Handbook A Guide For System Life Cycle Processes And Activities, INCOSE-TP-2003-002-03.2, International Council on Systems Engineering (INCOSE), 7670 Opportunity Rd, Suite 220, San Diego, CA 92111-2222, January 2010.
23. Laporte, C.Y., April, A., Renault, A., Applying ISO/IEC JTC 1/SC7 Software Engineering Standards in Very Small Enterprises, CrossTalk - The Journal of Defense Software Engineering, February 2007, Vol. 20, No 2, pp 29-30

NOTES

1. ISO/IEC JTC1/SC7 stands for International Organization for Standardization/International Electrotechnical Commission Joint Technical Committee 1/Sub Committee 7.
2. A previous Crosstalk article describes the establishment of Working Group 24 [7] [23].

Upcoming Events

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

Systems Engineering, Test and Evaluation Conference
29 April – 1 May 2013
Canberra, Australia
http://sapmea.asn.au/conventions/sete2013

IBM Edge 2013
10-14 Jun 2013
Las Vegas, NV
http://www.ibm.com/edge

23rd Annual INCOSE International Symposium
24-27 Jun 2013
Philadelphia, PA
http://www.incose.org/symp2013

Software Assurance Working Group Sessions
25-27 Jun 2013
McLean, VA
https://buildsecurityin.us-cert.gov/bsi/events.html

Star Trek, Split Infinitives, and Agile Programming

"Space … the final frontier. These are the voyages of the Starship Enterprise. Its five-year mission: to explore strange new worlds, to seek out new life and new civilizations, to boldly go where no man has gone before."

If you are a reader of CrossTalk, it is a pretty safe bet that you recognize the quote above.

Did you notice the problem lurking in the above quote that has vexed English speakers and writers for almost 200 years? Yes, it is the dreaded split infinitive. It should be "…to go boldly…." The split infinitive is still the subject of disagreement among native English speakers as to whether it is grammatically correct. According to Wikipedia, no other grammatical issue has so divided English speakers since the 19th century, when the split infinitive was declared to be incorrect.

As part of my profession, I write articles—and have done so for over 25 years. I consider myself a pretty decent wordsmith. I have taken several college courses in Technical Writing. Back in 2007, an associate editor at a journal noticed a split infinitive in one of my drafts, and sent me a note jokingly saying, "You should know better." The associate editor was "old school." I, on the other hand, thought nothing about it—because I do not really feel that grammatical rules that originally applied to Latin grammar 2,000 years ago should dictate how I write English today. Modern English usage guides seem to agree—most have dropped their opposition to the split infinitive. Split infinitives were considered wrong 50 years ago, but today they are accepted as correct.

Which brings me to agile programming. I hate to admit it—I was "old school." If the term "2167A" does not make you tremble, you were not developing software 25 years ago. 2167A was a DoD standard, published in 1988. It established, "uniform requirements for software development that are applicable throughout the system lifecycle."

One criticism of 2167A was that it was biased toward the waterfall model. While it did include, "the contractor is responsible for selecting software development methods (for example, rapid prototyping)," 2167A also required "formal reviews and audits" that seemed to lock the vendor into designing and documenting the system before any implementation began. 2167A focused on design documents, rather than the use of accepted CASE tools. Vendors would often use CASE tools to design the software, but then be forced to write 2167A-required documents to describe the CASE-formatted data. The result was that as the system evolved, the CASE tool products could be updated easily, but not the documents. Over time, the design documents required by 2167A became obsolete (assuming they were ever of any real use).

Software development methodologies evolved. In the 1990s we had the CMM®. I was and am a CMM proponent, but let me quote from Wikipedia, "The model was originally intended to evaluate the ability of government contractors to perform a software project. It has been used for and may be suited to that purpose, but critics pointed out that process maturity according to the CMM was not necessarily mandatory for successful software development. Real-life examples where the CMM was arguably irrelevant to successful software development include many shrinkwrap companies (also called commercial-off-the-shelf or "COTS" firms or software package firms). Such firms would have included Claris, Apple, Symantec, Microsoft, and Lotus. Though these companies have successfully developed their software, they have not considered, defined, or managed their processes as the CMM described as level 3 or above, and so would have fitted level 1 or 2 of the model."

In other words, companies using CMM could develop good software, but good software did not depend on the company using CMM.

You see, the role and nature of software development keeps evolving. Early languages had different rules, different purposes, and different focuses. It is hard to imagine agile development in COBOL and Fortran. Modern languages, like C#, Ruby, Perl, and Java have a very close relationship with component libraries, system-supplied support libraries, and higher-level abstractions. It becomes more and more possible to quickly develop something that works—and still have quality software. As long as the software meets good software engineering standards—reliable, understandable, modifiable, and efficient—agile development can get the job done.

Just like Latin grammar rules, perhaps older software development rules are sometimes a bit less relevant nowadays. Back in the 90s, the mere mention of agile sent large-scale software developers screaming "disbeliever" and running for holy water to throw on the heretics. 20 years later, it is a generally accepted way to perform large-scale system development and achieve quality results.

It is all about meeting customer needs and producing quality software. I am not saying the old ways will not work anymore—I am just saying the new ways work, too.

It is the 2010s. Maybe it is time for you "to boldly go" where you have never gone before. Split infinitive or not.

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

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

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

CrossTalk thanks the above organizations for providing their support.