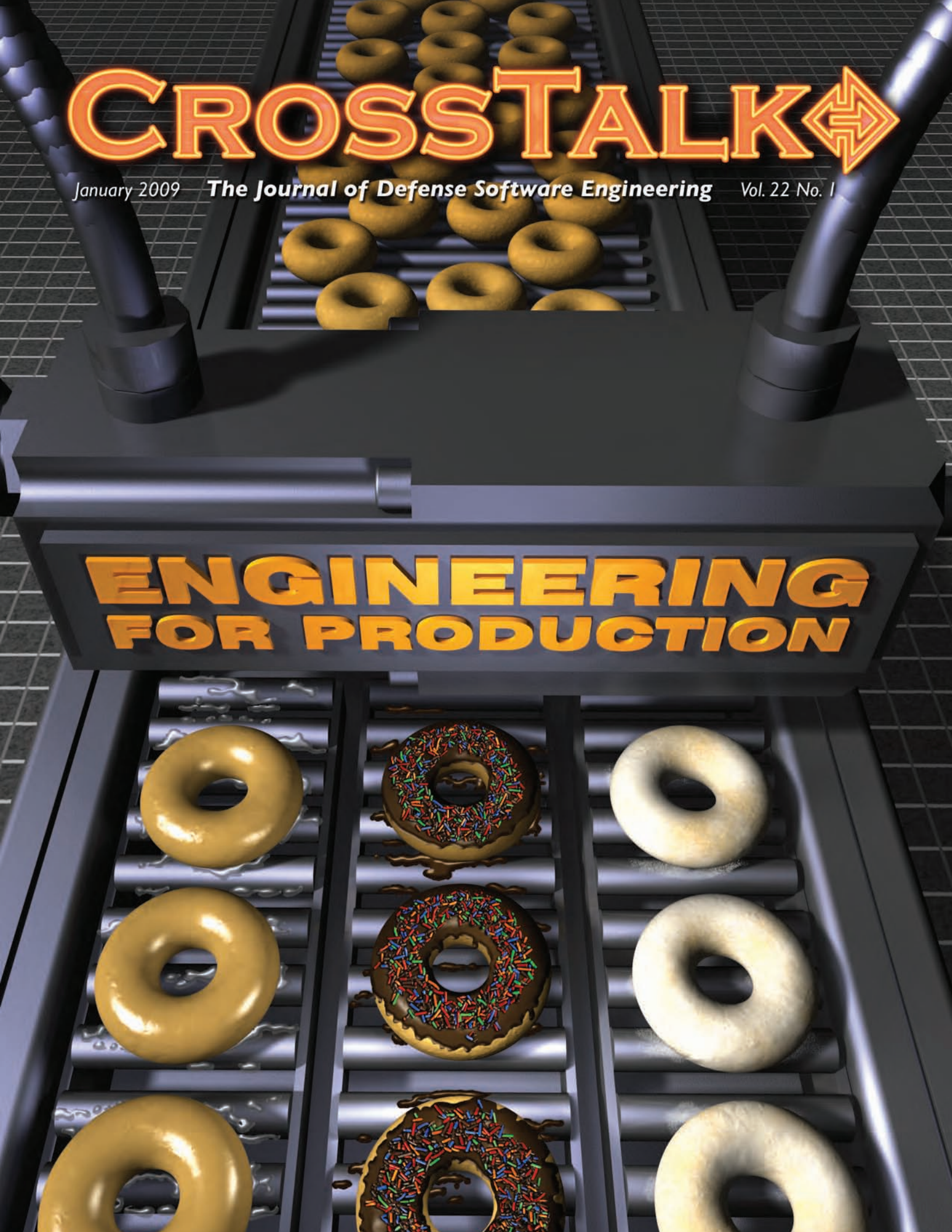

CROSSTALK
January 2009 The Journal of Defense Software Engineering Vol. 22 No. 1
ENGINEERING
FOR PRODUCTION

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JAN 2009		**00-00-2009 to 00-00-2009**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Crosstalk, The Journal of Defense Software Engineering. Volume 22, Number 1, January 2009	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/MXDEA,6022 Fir Avenue,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)**	**32**	

Policies, News, and Updates

Engineering for Production

Software Engineering Technology

Open Forum

On The Cover

Cover Design by Kent Bingham

Additional art services provided by Janna Jensen

Departments

CrossTalk

OSD (AT&L)	*Kristen Baldwin*
NAVAIR	*Joan Johnson*
309 SMXG	*Karl Rogers*
DHS	*Joe Jarzombek*
Managing Director	*Brent Baxter*
Publisher	*Kasey Thompson*
Managing Editor	*Drew Brown*
Associate Editor	*Chelene Fortier-Lozancich*
Publishing Coordinator	*Nicole Kentta*
Phone	(801) 775-5555
E-mail	stsc.customerservice@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/crosstalk

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

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

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 19.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CrossTalk does not pay for submissions. 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 author and their organizations.

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

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

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

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

Engineering for Production

Software architecture is gaining traction within the DoD acquisition community as a viable and necessary part of the acquisition and development process. Software product lines (SPLs), when integrated through a systematic approach, are the penultimate expression of sound software architectural technique. The ability to satisfy a wide array of needs with a single, configurable product is extremely powerful.

SPLs can save taxpayer money, quickly adopt next-generation technology, and turn a software support activity into its own program management entity. SPLs also have the ability to cross service and national borders, eliminate vendor-proprietary, stove-piped software solutions, and quickly adapt to new technology.

While some practitioners, developers, and managers see multiple benefits in this overarching approach, others may feel threatened. The basic question of how to retain technical and fiscal control of all aspects of their systems, while protecting the bottom line, is important. And can a project manager truly rely on this approach without threatening core capability or leaving their technology behind?

These concerns present us practitioners with many challenges. We must build trust with program managers, proving over and over that their needs take a back seat to no one, and their dollars are well and properly spent. We must demonstrate that their trust buys them more capabilities and higher quality than their original dollars could ever have afforded. We must develop incentive/reward approaches that encourage development and use of SPLs by developers, acquirers, and vendors. We must enable vendors to explore advanced development of implementations that propose and meet future needs, and empower them to participate in the maturation and expansion of the SPL's architecture and market.

The real issue is that every challenge must be met, and met concurrently. This means changing the acquisition culture so that SPLs and other software production efficiencies are the expectation, not the exception. In turn, the public gets the most for their hard-earned taxes, and the warfighter gets the highest quality, best of breed software capability possible.

No single one of these challenges is insurmountable. There are examples of SPLs having some success within the DoD, such as with the Tactical Control System, the Portable Flight Planning System, and the Joint Mission Planning System.

The articles in this issue of CrossTalk address an innovative look at a wide spectrum of possibilities to consider when focusing on SPLs. In *Production Planning for a Software Product Line*, Dr. Gary J. Chastek, Linda M. Northrop, and Dr. John D. McGregor present a three-step approach to taking an organization from their SPL goals to a comprehensive production plan. To better understand what project managers might face when transitioning to an SPL approach, read Dr. Paul Jensen's article, *Experiences With Software Product Line Development*, on the experiences at Overwatch Systems during their move to an SPL. Dr. Christof Ebert's article, *Software Product Management*, outlines the role of a software product manager and provides software product management best practices that will help in attaining market success.

There are also examinations of other ways to improve software production and overall effectiveness. In *"Spending" Efficiency to Go Faster*, Dr. Alistair Cockburn shows how software development projects can improve total system results by reducing the impact of "bottlenecks." Through interviews with practitioners, Dr. Mahmood Niazi identifies the critical barriers that impede Software Process Improvement programs and provides guidelines to help avoid those barriers in his article, *Software Process Improvement Implementation: Avoiding Critical Barriers*. And if your software organization needs to change course, you'll want to read Esther Derby's article, *Three Encouraging Developments in Software Management*, which examines the benefits of evidence-based management, Lean principles, and Agile methods.

I know you will enjoy the valuable insights that this issue of CrossTalk brings to the challenges and possibilities of transitioning to SPLs, as well as to other software practices.

David Curry
NAVAIR Co-Sponsor

Announcing CrossTalk's Co-Sponsor Team for 2009

Kasey Thompson
CrossTalk

First of all, I would like to once again express sincere thanks to the 2008 CrossTalk *co-sponsors. Simply put,* CrossTalk *would not exist without them and their generous financial support. As Publisher, I receive countless kudos—via e-mail and the phone—expressing appreciation for an article or issue focus that contributed to individual or organizational success. These compliments really belong to the co-sponsors, who spark countless themes and help bring us the best authors in defense software engineering. Likewise, it is my pleasure to introduce* CrossTalk*'s 2009 co-sponsor team and offer profound gratitude for their continued support and commitment to this journal. I know firsthand of their vision, caring, and dedication to their industry and it is manifested through support of* CrossTalk*. Each co-sponsor and their organization will assist our staff by lending us their inexhaustible experience in engineering, systems, security, acquisition, tools, processes, models, infrastructure, people, and of course, software. Co-sponsor team members are identified in this section with a description of their organization. Please look for their contributions each month in our From the Sponsor column, found on page 3. Their organizations will also be highlighted on the back cover of each issue of* CrossTalk.

Kristen Baldwin, Office of the Secretary of Defense – Acquisition, Technology and Logistics

Software Engineering and Systems Assurance is the staff agent responsible for all matters relating to DoD software engineering, systems assurance, and system of systems (SoS) engineering. Organizational focus areas include policy, guidance, human capital, acquisition program support, software acquisition management and engineering, software and systems engineering integration, SoS enablers and best practices, engineering for system assurance, and government-industry collaboration. See <www.acq.osd.mil/sse/ssa> for more information.

Joan Johnson, NAVAIR, Systems Engineering Department – Director, Software Engineering

The Naval Air Systems Command (NAVAIR) has three Strategic Priorities through which it produces tangible, external results for the Sailor and the Marine. First are its People that we develop and provide the tools, infrastructure, and processes needed to do their work effectively. Next is Current Readiness that delivers NAVAL aviation units ready for tasking with the right capability, at the right time, and the right cost. Finally is Future Capability in the delivery of new aircraft, weapons, and systems on time and within budget that meets Fleet needs and provides a technological edge over our adversaries. See <www.navair.navy.mil> for more information.

Karl Rogers, 309 SMXG Acting Director

The 309th Software Maintenance Group at the Ogden-Air Logistics Center is a recognized world leader in cradle-to-grave systems support, encompassing hardware engineering, software engineering, systems engineering, data management, consulting, and much more. The division is a Software Engineering Institute Software Capability Maturity Model® (CMM®) Integration Level 5 organization with Team Software Process℠ engineers. Their accreditations also include AS 9100 and ISO 9000. See <www.mas.hill.af.mil> for more information.

Joe Jarzombek, Department of Homeland Security – Director of Software Assurance

The DHS National Cyber Security Division serves as a focal point for software assurance (SwA), facilitating national public-private efforts to promulgate best practices and methodologies that promote integrity, security, and reliability in software development and acquisition. Collaborative efforts of the SwA community have produced several publicly available online resources. For more information, see the Build Security In Web site <https://buildsecurityin.us-cert.gov> and the SwA Community Resources and Information Clearinghouse <https://buildsecurityin.us-cert.gov/swa> to provide coverage of topics relevant to the broader stakeholder community.

Want to Become a Co-Sponsor?

CrossTalk co-sponsors enjoy many benefits such as inclusion of a page-long co-sponsor's note, placement of their organization's logo on the back cover for 12 issues, placement of the Director's name and organization on each issues' masthead, special sponsorship references in various issues, the ability to provide authors from within their community in regard to their sponsored issue, and online placement on the CrossTalk Web site.

CrossTalk co-sponsors are also invited each year to provide direction for future CrossTalk themes and feedback from the software defense community at large. Co-sponsors are also invited to participate in an annual meeting held during the Systems and Software Technology Conference to discuss emerging needs, trends, difficulties, and opportunities which CrossTalk may address in an effort to best serve its readers.

CrossTalk welcomes queries regarding potential sponsorship throughout the year. For more information about becoming a CrossTalk co-sponsor, please contact Kasey Thompson at (801) 586-1037 or <kasey.thompson@hill.af.mil>.

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

℠ Team Software Process are service marks of Carnegie Mellon University.

The 2009 CrossTalk Editorial Board

Kasey Thompson
CrossTalk

CrossTalk proudly presents the 2009 CrossTalk Editorial Board. Each article submitted to CrossTalk is reviewed by two technical reviewers from the list below. Their insights improve the readability and usefulness of the articles that are published in CrossTalk. Most reviewers listed have graciously offered their own time to support CrossTalk's technical review process. We give a very special thanks to all those participating in our 2009 CrossTalk Editorial Board.

Wayne Abba	Abba Consulting
COL Ken Alford, Ph.D.	Brigham Young University
Bruce Allgood	Electronic Systems Center Engineering Directorate
Brent Baxter	Software Technology Support Center
Jim Belford	OO-ALC Engineering Directorate
Gene Bingue	U.S. Navy
Lt. Col. Christopher A. Bohn, Ph.D.	HQ Air Force Special Operations Command
Mark Cain	309th Software Maintenance Group
Dr. Alistair Cockburn	Humans and Technology
Richard Conn	Retired (formerly with Microsoft)
Dr. David A. Cook	The AEgis Technologies Group, Inc.
Rushby Craig	309th Software Maintenance Group
Les Dupaix	Software Technology Support Center
Monika Fast	309th Software Maintenance Group
Robert W. Ferguson	Software Engineering Institute
Dr. John A. "Drew" Hamilton Jr.	Auburn University
Gary Hebert	538th Aircraft Sustainment Group
Tony Henderson	Software Technology Support Center
Lt. Col. Brian Hermann, Ph.D.	Defense Information Systems Agency
Lt. Col. Marcus W. Hervey	Air Force Institute of Technology
Thayne Hill	Software Technology Support Center
George Jackelen, PMP	Software Consultants, Inc.
Deb Jacobs	Priority Technologies, Inc.
Dr. Randall Jensen	Software Technology Support Center
Alan C. Jost	Raytheon – Network Centric Systems
Daniel Keth	Software Technology Support Center
Paul Kimmerly	U.S. Marine Corps
Theron Leishman	Northrop Grumman
Glen L. Luke	309th Software Maintenance Group
Gabriel Mata	Software Technology Support Center
Jim McCurley	Software Engineering Institute
Paul McMahon	PEM Systems
Dr. Max H. Miller	Raytheon Integrated Defense Systems
Mark Nielson	Software Technology Support Center
Mike Olsem	Science Applications International Corporation
Glenn Palmer	L-3 Communications, Inc.
Doug J. Parsons	Army PEO Simulation, Training and Instrumentation
Tim Perkins	500 CBSS/GBLA (Atmospheric Early Warning System)
Gary A. Petersen	Arrowpoint Solutions, Inc.
Vern Phipps	SabiOso
David Putman	309th Software Maintenance Group
Kevin Richins	Arrowpoint Solutions, Inc.
Gordon Sleve	Robbins Gioia LLC
Larry Smith	Software Technology Support Center
Elizabeth Starrett	Software Technology Support Center
Tracy Stauder	309th Software Maintenance Group
COL John "Buck" Surdu, Ph.D.	Army Research, Development, and Engineering Control
Dr. Will Tracz	Lockheed Martin Integrated Systems and Solutions
Jim Van Buren	Charles Stark Draper Laboratory
David R. Webb	309th Software Maintenance Group
Mark Woolsey	Software Technology Support Center
David Zubrow	Software Engineering Institute

Production Planning for a Software Product Line

Dr. Gary J. Chastek and Linda M. Northrop
Software Engineering Institute

Dr. John D. McGregor
Clemson University

The goal of using a software product line (SPL) approach is to predictably develop multiple software-intensive systems (products) in an efficient, timely, and cost-effective manner that takes economic advantage of the features common to the products. Achieving this goal requires more than reusable (core) assets. It requires production planning that formulates a production strategy, devises a production method, and composes a production plan that is followed for each product. We present a three-step approach to production planning that guides an organization from the goals for the SPL to a comprehensive production plan.

The benefits of an effective production system are well understood. For example, in manufacturing, the Toyota Production System (TPS) produces high-quality, low-cost automobiles and is widely credited with providing Toyota with a significant competitive advantage. The TPS is based on a set of development rules [1], two of which are of particular relevance to SPLs:

- **Rule 1 – How People Work.** Tasks are rigorously specified yet the overall processes are highly flexible.
- **Rule 2 – How People Connect.** Customers and suppliers are connected directly and unambiguously, with defects and schedules handled rapidly.

Just as producing automobiles is Toyota's focus, producing software-intensive products is the main business of an SPL organization. An SPL is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [2]. This definition is consistent with the one traditionally given for any product line. But it adds more, putting constraints on the way in which the systems in an SPL are developed. Why? Because substantial production economies can be achieved when the systems in an SPL are developed from a common set of assets in a prescribed way, in contrast to being developed separately, from scratch, or in an arbitrary fashion. It is exactly these production economies that make the SPL approach succeed in ways that earlier forms of software reuse failed.

Building a new product (system) becomes more a matter of assembly or generation than one of creation; the predominant activity is integration rather than programming. For each SPL, there is a production plan that specifies the exact product-building approach. And software components are not the only thing being reused across products. Other core assets include requirements, architecture, designs, documentation, budgets and schedules, tools, process definitions, performance models, testing artifacts, and much more.

The organizations that we have studied[1] have achieved remarkable benefits that are aligned with commonly held business goals, including:

- Large-scale productivity gains.
- Decreased time-to-market.
- Increased product quality.
- Decreased product risk.
- Increased market agility.
- Increased customer satisfaction.
- More efficient use of human resources.
- The ability to effect mass customization.
- The ability to maintain market presence.
- The ability to sustain unprecedented growth.

These benefits give organizations a competitive advantage and are derived from the reuse of the core assets in a strategic and prescribed way.

Implied by the definition of an SPL are two coordinated roles in product production. The core asset developer creates reusable artifacts that will be used to build multiple products. The product developer assembles products from these core assets—not arbitrarily, but in a *prescribed way*. In [3], the authors surveyed a number of organizations and found that these two roles are often embodied in separate teams that are not co-located. The artifacts produced during production planning are intended to coordinate the interactions of these two roles.

The core assets are designed to be reused, and reused in particular ways that accommodate the variation that is inherent in separate products. Although the product developers construct the products, they are constrained in what they can do because of how the core assets are designed. Production planning provides the opportunity for technical planning on how the core assets should be designed to facilitate efficient product variation and integration [4]. The production plan ultimately describes how to use the core assets to build products.

These described TPS rules illustrate some of the implications of the SPL definition for product production. In a product line production system, the prescribed way is intended to rigorously specify the roles in the organization. This specification is actually broken into pieces that correspond to the individual core assets used to build the products. Each asset is accompanied by a description of how to use the asset in product building. This attached process is used to populate portions of the production process. As different products are defined and different assets are selected for inclusion in the products, different attached processes are included in the production plan and *how people work* is changed.

In an SPL organization, the core asset and product developers are connected by a delivery process in which core assets are made available to the product developers. This is developed from a common set of core assets from the SPL definition. An explicit feedback mechanism allows the product developers to inform the core asset developers about any defects or to request desired enhancements. This is a realization of Rule 2 of the TPS.

In the following sections, we provide an overview of our approach to production planning, list several documented production planning experiences in both industry and the DoD, and describe a production planning workshop we have used with industrial and DoD customers. Due to the size of the artifacts, readers should examine the production planning artifacts of the Software Engineering Institute's (SEI's) Pedagogical Product Line [5]. This

example SPL provides an extensive set of product line artifacts in addition to the production plan.

Approach

Our production planning technique involves three activities: context, prerequisites for planning, and planning activities [6]. Each activity produces an artifact that plays a specific role in the production system. We will describe the context in which the activities are conducted, the prerequisite actions that must occur prior to production planning, and then detail each of the three planning activities.

Context

Production planning occurs within the context of the SPL, the product line organization, and the narrower context defined by the product and production constraints. These constraints are identified during early product line analysis activities.

Activities such as scoping and market analysis identify production constraints. The structure of the organization can also impose constraints. The relationships among the customers, program offices, and contractors constrain production. These constraints include required divisions in process responsibilities brought on by the geographic distribution of personnel and legal divisions between the program office and the contractor.

Required properties of the products impose constraints. For example, the need for DO-178B certification imposes the requirement that the tools and processes used to produce the core assets and products be approved for this level of quality.

Prerequisites for Planning

Production planning begins early in the planning for the product line but it can only be completed when certain prerequisites have been met. The requirements for the production system are captured in a set of production scenarios. A production scenario takes the form shown in Table 1. Sufficient production scenarios are created to cover the various ways in which products will be produced.

Planning Activities

The planning activities form a logical sequence that move planners from determining the goals of the production system to identifying the specifics of the production schedule for each product.

Defining the Production Strategy

The production strategy is a high-level statement of how the organization expects to achieve the goals of the product line. The breadth and longevity of a product line requires a goal-driven approach to keep the organization focused. The technique for defining the production strategy begins with the business goals of the product line organization.

Source of Stimulus	Who or what is initiating product production.
Stimulus	The event or action that initiates product development.
Environment	The state of the production environment of the product line at the time of this scenario (e.g., all core assets are completed and available for use).
Artifact	The production system artifact can be a product or a core asset.
Response	How the production system responds to the request to produce a specific product. For example, how long will it take to produce this product?
Response Measure	The measure may be calendar days from purchase contract to deployment, cost in dollars or days of effort, etc.

Table 1: *Production Scenario*

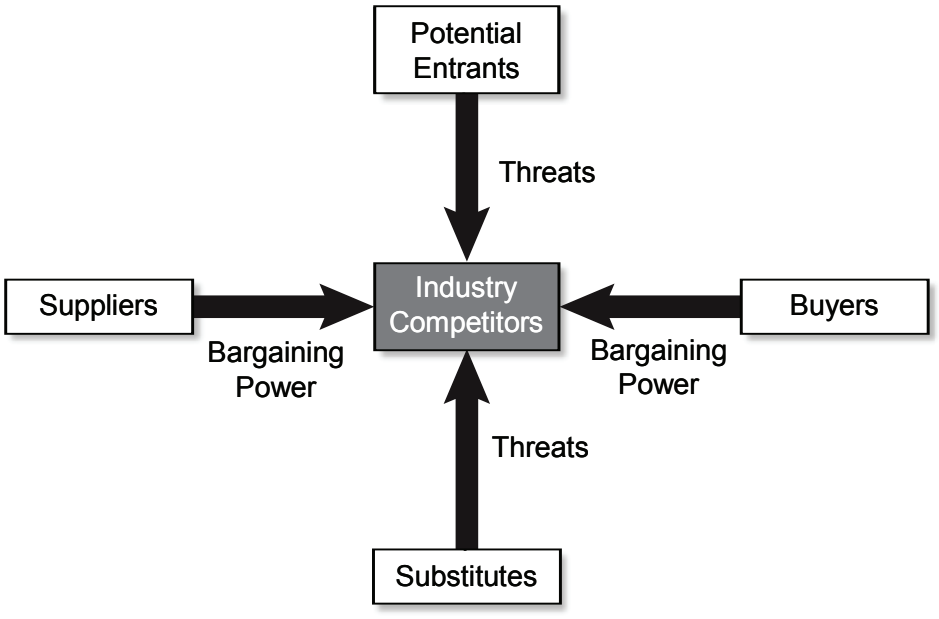
The SEI *What to Build* pattern [2] focuses on defining which products are part of the product line, its scope, and developing a business case that justifies the investment in the product line. The business case is predicated on the organization's business goals and identifies the factors that are critical to the success of the product line.

We use Porter's Five Forces Model [7], illustrated in Figure 1, and the critical factors output from application of the *What to Build* pattern, to identify the strategic actions that will be the basis of the production strategy.

The boxes represent the market forces and the arrows represent threats and power. For example, potential entrants into a market represent a threat while buyers use their bargaining power to obtain discounts or improved products. Here we provide a brief description of each force:

- **Potential Entrants.** How can we raise the cost to others of entering the market by the means we use to produce the products? In an SPL context, this usually results in a strategic action to automate as much as possible, amortizing automation costs over the set of products. This in turn increases the cost of entry for potential competitors. A flexible production method can also respond to the needs of customers faster than product producers who have not yet entered the market.
- **Substitutes.** How can we differentiate

Figure 1: *Porter's Five Forces Model*



our product from the substitutes through the means of production? The economies of scale of the product line support a strategic action to lower prices while the economies of scope support an increase in features to resolve the threat of substitutes.

- **Buyers.** How can we better respond to buyers' requests through attributes of the production process? One strategy action is to adopt short iterations that provide enhanced functionality quickly. Another is to provide buyers with access to the status of defect fixes so they can track those that are important to them.
- **Suppliers.** How can we lower the prices we pay suppliers by the production techniques we use? The use of open source software is one strategic action. Another is to take advantage of the economies of scale of the product line to negotiate lower license fees for specific components.
- **Industry Competitors.** How can we gain advantage over the competition by different choices of production techniques? One strategic action is building multiple versions of products simultaneously. By establishing collaborative production arrangements with suppliers where we obtain early copies of future versions of their components, our products can be released much sooner after the release of the new components.

The strategy that results from this activity links the business goals of the product line to a first, high-level statement about how products will be produced. The strategy provides an essential input into the development of the production method. A detailed description of production strategy development can be found in [8].

The DoD context adds a layer to the usual customer/supplier relationship. The customer and program office each have goals for the product line and some of these goals can be achieved through the appropriate production techniques. The production strategy is ultimately the responsibility of the contractor who will build the product line, but it must encompass the goals of the customer and program office.

Engineering the Production Method

The production method [9] bridges the gap between the production strategy and production plan to provide a comprehensive view of the entire SPL development. While the production method is intended to describe how to produce a product, it also defines constraints on how the core asset developers can design their assets. The method becomes the vehicle for coordination between the core asset and product development teams.

"The production method bridges the gap between the production strategy and production plan to provide a comprehensive plan of the entire SPL development."

The method encompasses the processes, tools, and models needed to completely describe a development effort. For example, the production method, in response to the production strategy, might adopt an Agile process model for productbuilding teams. The method would describe the roles and tasks for customers or customer surrogates and development team members. The method would define work products such as user stories, unit tests, and a software architecture, and assigns responsibility for their creation to specific roles.

The scope of the method varies from one organization to another. The method incorporates the processes and associated tools and models for building products, but the method may be expanded to include processes for specifying products and for product deployment. In some product line organizations, the production method may also include management activities related to estimating and scheduling production.

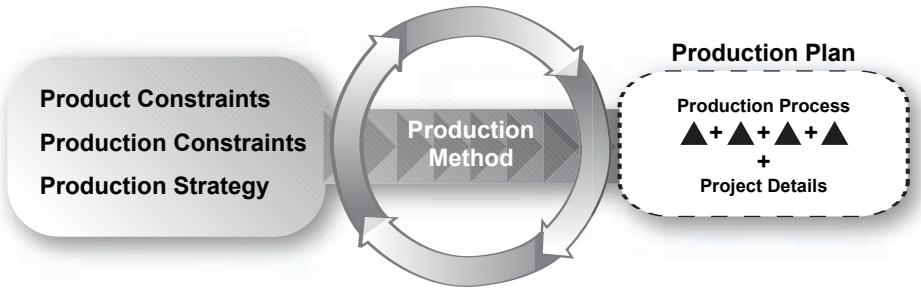
Development of the production method usually begins either with the single system development method that is in use as the product line organization is formed or with a standard software development method. Method engineering techniques [10] are used to elaborate that method into the full production method that addresses the scale and variability of the product line.

Developing the Production Plan

The production plan (shown in Figure 2) is the product builder's guide. It prescribes how products are produced from the core assets. It includes a process to be used for building products (the production process) and lays out the project details to enable execution and management of the process. The production plan is structured around the product building process defined in the production method. Just like a product specification, the production plan includes variation points; these include the variation points in the product specification as well as points related to the potential variations in the production system.

The production plan gives a step-by-step description of how to build a product. The initial description is high-level, but it is particularized as choices are made at variation points and core assets are selected to implement those choices. Each core asset is accompanied by an *attached process*. The attached process for an asset is the user's guide for that asset. The attached process solves the common software reuse problem of the learning curve for a new asset. The attached process for a core asset gives a step-by-step process for using the asset and should provide sufficiently detailed information to combat any difficulties in understanding how the asset works. The attached process for a selected asset is added into the product's production process when that asset is selected to be used to build the product. By making core asset selections in the prescribed order and adding the attached process for each asset as it is selected, a fully instantiated plan is developed just in time to guide the product developer [11, 12].

Figure 2: *Production Plan*



Production Planning Experiences

Our experience has shown the value of production planning:

- A survey was conducted of SPL Hall of Fame members to capture their experiences with production planning.

The full results of that survey are available in [3]. The most important result (for the purposes of this article) is that production planning was found to have a positive effect on the success of the product line. Organizations that did not sufficiently focus on production planning wished they had. Cummins, Inc. included more robust architecture and production planning practices in its second generation product line [13].

- Paul Jensen describes Overwatch's experience with production planning in a DoD context [14]. This experience illustrated the need to plan for production as early as possible. An attempt to change tools in the midst of core asset development was difficult and resulted in inadequate tool support.
- In [15], the authors describe a *user guide* that is essentially a production plan at Rolls Royce. The user's guide, for the core asset base, describes how to use the core assets to build products. It was built as a core asset and delivered with the core asset base.
- In [16], the authors provide a production planning technique that relies on the feature model. Production planning was decomposed into planning about how to include each feature in a product. By maintaining this traceability, the product line production plan is more easily transformed into the product-specific production plan. As features are selected during product definition, the production plan is composed.

The SEI developed a Production Planning Workshop [17] in response to requests for assistance on initiating a production planning capability. The workshop is an intensive planning session that is intended to expedite an organization's planning, providing a two-day introduction to the production planning process. Our experience with the introductory workshop has shown us that these two days are very useful in accelerating the production planning exercise.

Summary

A robust production planning technique—such as the one we have described—produces a production plan that is actionable and evolvable. The organization extracts production goals from the overall product line goals, creates a production strategy that resolves the competitive forces through the means of production, elaborates the strategy into a set of mutually consistent processes, tools, and models, and finally operationalizes the method in a detailed production plan. This sequence of successive refinements ensures that the production plan contains sufficient detail for core asset and product builders to accomplish their tasks predictably and efficiently. The traceability provided by this approach ensures that changes to product line goals or the discovery of additional production constraints can be propagated through the artifacts promptly.

A production system does not come free with reusable components, services, or even a product line architecture. A production plan is necessary. Developing and maintaining an effective and efficient production system is critical to the success of a product line.◆

References

1. TPS. "Toyota Production System Terms." Toyota Motor Manufacturing Kentucky, Inc. 2008 <www.toyotageorgetown.com/terms.asp>.
2. Clements, Paul, and Linda Northrop. Software Product Lines: Practices and Patterns. Boston: Addison-Wesley Publishers, 2002.
3. Chastek, Gary, Patrick Donohoe, and John D. McGregor. "A Study of Product Production in Software Product Lines." SEI, Carnegie Mellon University. Technical Note CMU/SEI-2004-TN-012. Mar. 2004 <www.sei.cmu.edu/pub/documents/04.reports/pdf/04tn012.pdf>.
4. Northrop, Linda, and Paul Clements. "A Framework for Software Product Line Practice, Ver. 5.0." SEI, Carnegie Mellon University. 2007 <www.sei.cmu.edu/productlines/framework.html>.
5. SEI. "Arcade Game Maker – Pedagogical Product Line." SEI, Carnegie Mellon University. 2008 <www.sei.cmu.edu/productlines/ppl/>.
6. Chastek, Gary, Patrick Donohoe, and John D. McGregor. A Production System for Software Product Lines. Proc. of the 11th Annual Software Product Line Conference. Kyoto, Japan, 2007: 117-128.
7. Porter, Michael E. "Competitive Advantage: Creating and Sustaining Superior Performance." Free Press, 1998.
8. Chastek, Gary, and John D. McGregor. "Formulation of a Production Strategy for a Software Product Line." SEI, Carnegie Mellon University. Technical Report CMU/SEI-2008-TN-023.
9. Chastek, Gary, Patrick Donohoe, and John D. McGregor. "Applying Goal-Driven Method Engineering to Product Production in a Software Product Line." SEI, Carnegie Mellon University. Technical Report to appear 2009.
10. Henderson-Sellers, Brian. "Method Engineering for OO Systems Development." Communications of the ACM 46 (Oct. 2003): 73-78. <http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=944242>.
11. Chastek, Gary, and John D. McGregor. "Guidelines for Developing a Product Line Production Plan." SEI, Carnegie Mellon University. Technical Report CMU/SEI-2002-TR-006. June 2002 <www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr006.pdf>.
12. Chastek, Gary, Paul Donohoe, and John D. McGregor. "Product Line Production Planning for the Home Integration System Example." SEI, Carnegie Mellon University. Technical Note CMU/ SEI-2002-TN-029. Sept. 2002 <www.sei.cmu.edu/pub/documents/02.reports/pdf/02tn029.pdf>.
13. Dager, Jim. Nomination presentation at Product Line Hall of Fame Session. 12th Annual Software Product Line Conference. Limerick, Ireland, Sept. 2008.
14. Jensen, Paul. Experiences With Product Line Development of Multi-Discipline Analysis Software at Overwatch Textron Systems. Proc. of the 11th Annual Software Product Line Conference. Kyoto, Japan, 2007.
15. Habli, Ibrahim, Tim Kelly, and Ian Hopkins. Challenges of Establishing a Software Product Line for an Aerospace Engine Monitoring System. Proc. of the 11th Annual Software Product Line Conference. Kyoto, Japan, 2007.
16. Lee, Jaejoon, K.C. Kang, and Sajoong Kim. A Feature-Based Approach to Product Line Production Planning. Proc. of the Third Software Product Line Conference. Boston, 2004: 183-196.
17. SEI. Software Product Lines. "Production Plans for Software Production Lines." 2008 <www.sei.cmu.edu/productlines/production_plan.html>.

Note

1. The SEI has published several detailed case studies of successful product line organizations and the benefits they have enjoyed. You can find these case studies in [2] and on the Web at <www.sei.cmu.edu/productlines/spl_case_studies.html>. You can also find references to product line efforts at <www.sei.cmu.edu/productlines/plp_hof.html> and <www.sei.cmu.edu/productlines/plp_catalog.html>.

About the Authors

Gary J. Chastek, Ph.D., is a senior member of the technical staff at the SEI in the SPL Initiative. Before joining the SEI, Chastek designed and implemented Ada compilers. Chastek's current research interests include production planning, variability management, and the use of open source development techniques in SPLs. He received his doctorate in computer science from the University of Pittsburgh in 1983.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213-2612
Phone: (412) 268-2826
E-mail: gjc@sei.cmu.edu

Linda M. Northrop is director of the Research, Technology, and System Solutions Program at the SEI, where she leads work in software architecture, SPLs, model-based engineering, integration of software-intensive systems, predictable software construction, and ultra-large-scale systems. She is co-author of "Software Product Lines: Practices and Patterns" and "Ultra-Large-Scale Systems: The Software Challenge of the Future."

SEI
4500 Fifth AVE
Pittsburgh, PA 15213-2612
Phone: (412) 268-7638
E-mail: lmn@sei.cmu.edu

John D. McGregor, Ph.D., is an associate professor of computer science at Clemson University, a visiting scientist at the SEI, and a partner in Luminary Software, a software engineering consulting firm. His research interests include SPLs, model-driven development, and component-based software engineering. McGregor's latest book is "A Practical Guide to Testing Object-Oriented Software."

School of Computing
Clemson University
Clemson, SC 29634
Phone: (864) 656-5859
E-mail: johnmc@lumsoft.com

Web Sites

Software and Systems Process Improvement Network (SPIN)

www.sei.cmu.edu/collaborating/spins

A SPIN is an organization of professionals in a given geographical area who are interested in software and systems process improvement, and this Web site is the place to sign up. Joining a SPIN recognizes a commitment and loyalty to improving the state of software and systems engineering, as well as placing one in contact with a network of experts within their community. It is a practical forum for the interchange of ideas, information, and mutual support. Each regional SPIN is slightly different, based on the vision of the founders and the needs of the community. SPINs are made up of professionals from all sectors—industry, government, and academia (including students)—and include defense contractors, professional organizations, and independent consultants.

Software Product Lines (SPLs)

www.softwareproductlines.com

Devoted to the community of software engineers and managers interested in using SPL approaches to develop their software, the goal of this site is to provide software developers, product managers, and development managers with practical information on SPL issues, ranging from introductory concepts to advanced techniques. Learn about SPL concepts, the first steps for using SPL approaches, the benefits that may help convince bosses to use SPL, success stories, expert perspectives, and resources to use for learning more about SPL.

The Goldratt Institute

www.goldratt.com

What should an organization do when confronted with low overall performance results, difficulties securing or maintaining a strategic advantage in the marketplace, financial hardships, seemingly constant firefighting, poor customer service, and chronic conflicts between people? Utilizing the Theory of Constraints (TOC) may help. The Goldratt Institute, birthplace of the TOC, is a leading provider of TOC expertise, development, implementation, and education. Their approach begins with the development of corporate strategy and fans out to all operational aspects of a given organization, tightly integrating the strengths of Lean Six Sigma into an overarching TOC-based solution. Once the barriers that block parts of an organization from working together as an integrated system are removed, the result is significant and sustainable improvement in each problem area. This Web site provides robust, customizable processes, expert-level training and certification, technical support, mentorship, training materials, and planning in support of all TOC practices.

Validating Java for Safety-Critical Applications

http://javolution.org/doc/Man33955.pdf

With the real-time extensions, Java can now be used for safety-critical systems. It is therefore imperative to ensure that virtual machine implementations not only conform to the Real-Time Specification for Java (RTSJ) but also that efficiency and predictability are up to a certain standard. In particular, if the overhead incurred by RTSJ implementations are beyond a certain threshold, they may not suitable for safety-critical systems. With this in mind, the Web site outlines the development and maintenance of a test suite that addresses conformance as well as performance, and proved to be extremely useful in the critical process of selecting the Java Virtual Machine Platform.

Experiences With Software Product Line Development

Dr. Paul Jensen
Overwatch Systems

Overwatch Systems recently transitioned to a software product line (SPL) approach. Using its SPL, Overwatch Systems provides both software products and custom software system development in the domain of intelligence planning, collection, and analysis to the U.S. DoD and intelligence community. This article describes the approach taken in Overwatch Systems' transition, describes the product line architecture that is a key to the Overwatch Intelligence Center (OIC) SPL, and provides the lessons learned during the transition.

Overwatch Systems focuses on the development and fielding of multi-discipline data analysis software systems. Areas of expertise include data fusion, all-source analysis, signal intelligence acquisition and analysis, sensor network technology, and visualization. Starting in 2003 and continuing through 2007, the company transitioned to an SPL approach, producing the OIC product line, of which all new applications and systems are members. Multiple systems have been fielded as members, including a Signal Intelligence (SIGINT) collection and analysis system and an all-source analysis system.

During the Overwatch Systems transition, the company desired more information related to the application of SPL practices in business environments similar to our own: fulfilling government contracts by delivering members of an SPL. The goal of this article is to make such information available so that introducing this approach can be easier in the future.

Software Product Lines

As defined by the Software Engineering Institute (SEI), an SPL is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular mission and that are developed from a common set of core assets in a prescribed way [1]. In other words, an SPL consists of a family of software systems that have some common functionality and some variable functionality. To take advantage of the common functionality, reusable assets (referred to as core assets) are developed, which can be reused by different members of the family. Following an SPL approach involves not only developing core assets but also focusing on the systematic production and delivery of the different member systems (or variants) of the SPL family.

OIC Background

In 2003, Overwatch Systems was in a transition period. In previous years, a large portion of the business was dedicated to developing a large all-source analysis system for a single customer. This system consisted of approximately 10 million lines of code (LOC) and hundreds of components. To produce solutions for new customers, this large analysis system was cloned, modified, and extended to meet new requirements. Often, this approach led to difficulties caused by the complexity and dependencies encountered while adapting a legacy system to a new mission.

"Following an SPL approach involves not only developing core assets but also focusing on the systematic production and delivery of the different member systems of the SPL family."

Simultaneously, Overwatch Systems was beginning a large multi-year effort to produce situation understanding and data fusion capabilities for a second large customer. Although rigid constraints were imposed on the architecture and development of this software, the company made a commitment to create shared value for both of the current customers through the reuse of key components.

Transition to the SPL Approach

Preliminary Steps

Based on a clear business case, the decision to transition to an SPL approach was made by the CEO of the company. From the beginning, the general manager, chief architect, and vice president of engineering were the champions of the product line vision. Having strong support was critical to the successful SPL creation and operation.

Transitioning affects all parts of a business and introduces many risks. Clements and Northrop [1] identify 29 business practice areas with activities that are essential for product line success (e.g., architecture definition, process definition, funding, etc.). These practice areas cover all aspects of the business including software engineering, technical management, and organizational management. During this preliminary transition, a particular focus was put on the practice areas of mining existing assets and training. Although these two practice areas were critical to the transition, a lack of focus on other practice areas, such as architecture definition, funding, and structuring the organization, caused many problems that are discussed later in this article.

To reduce the risks, an incremental transition plan was created. This approach introduced product line concepts to various practice areas incrementally and relied on the creation of core assets to be realized as part of ongoing development activities. A specific customer system was targeted to be the first member of the SPL. This target provided motivation by creating a sense of urgency and forcing employees to focus on the product line transition instead of focusing only on normal daily issues.

The transition plan was derived from the SPL Factory Pattern, which is a composite pattern for an entire product line organization [1]. Activities that took place in this preliminary stage included scope definition, market analysis, a technical probe, training on product line concepts, and a determination of funding and organizational models. The technical probe was used to assess the current state of the organization in the context of a

product line transition and to track the progress of the transition over time. The training consisted of seminars for employees and required reading materials, such as [1]. The seminars covered product line concepts and how practice areas would be affected by the transition. The training was provided by the SEI's Paul Clements, the vice president of engineering, and the chief architect. This mixture of training provided both an expert view on SPLs and a description of product line practice areas specific to Overwatch Systems.

Based on our experience, the development of the funding model and organizational model are particularly important for success with an SPL approach. The organizational model that Overwatch Systems initially implemented was based on a division of domain engineering activity where reusable assets are created and customer-specific engineering activity in which the assets are assembled, customized, and extended. This model, as described in [2], appeared to be optimal. A domain engineering group was formed to produce reusable assets. Multiple customer-specific engineering groups would use those assets for customer system development. The funding for the domain engineering group was initially derived from internal Overwatch Systems funds. The plan was that these funds, occasionally augmented with funds from customer projects, would fully support the employees required for domain engineering.

Two major problems were encountered in operations that caused the company to eventually change this funding and organizational model. The first problem was that difficulties were encountered in funding a domain engineering group at a constant rate. Internal funds were insufficient to pay for a majority of the core asset development that was required. Fluctuations in customer project funding caused unmanageable fluctuations in the size of the domain engineering group. The second problem was that experts in certain technical areas were continuously pulled from the domain engineering group to the customer-specific engineering group to fill critical gaps.

In the second year of the transition, Overwatch Systems changed its organizational and funding model in order to address these problems. The company transitioned to an organizational model described as *mixed responsibility* [2], in which customer-specific engineering groups shared responsibility for developing reusable core assets. The lesson learned from this change is to ensure, through analysis and accurate estimation, that the funding and organizational model chosen for an SPL approach aligns with the demands and characteristics of one's business.

Starting Out

The initial domain engineering group consisted of eight engineers and would grow to more than 25 before the organization transition. The domain engineering group's mission was to create the necessary number of core assets in a new architecture that would allow the company to begin making systems that are members of the SPL.

> *"The initial SPL architecture was developed and focused primarily on the dependencies between components in the product line, emphasizing specific rules that governed what types of dependencies were allowed between product line assets."*

The initial SPL architecture was developed and focused primarily on the dependencies between components in the product line, emphasizing specific rules that governed what types of dependencies were allowed between product line assets. The architecture neglected to address issues related to the product engineering or assembly aspects of the products and did not define much of the infrastructure that, as the company discovered later, is needed to operate a product line efficiently.

Drawing from a legacy software baseline consisting of approximately 10 million LOC, the core asset mining process began. Several issues were encountered and had to be overcome. The company lacked experience in performing domain analysis, which is the process of determining the commonality and variability across all current customers and potential customers. Consequently, domain analysis was not performed adequately and the deciding factor in determining what assets to mine and what variations to implement became the requirements of the first customer system to use the product line. Domain analysis processes had to be determined, tested, and applied to overcome this issue.

The architecture and infrastructure developed were not sufficiently designed to support an SPL. Insufficient thought was put into determining how the core assets would be assembled and how the architecture could exploit commonality. As the core asset base expanded, many cases were observed where commonality was not being taken advantage of and software components were not well integrated. To address this problem, a revised architecture was created later.

The company did not possess expertise with variations in non-software core assets (e.g., related artifacts such as requirements and test procedures). The mining process started before the processes and tool modifications needed to support non-software core assets were executed. The result was an inadequate representation of variation and commonality, and a lack of variation dependency in these artifacts.

Despite these difficulties, approximately 4.5 million LOC were mined and new software components were developed, resulting in approximately 200 software core assets. These core assets provided a range of capabilities in the all-source and SIGINT intelligence domains, including data ingestion, management, processing, fusion, as well as geospatial, temporal, and relational visualization. In 2005, the first customer system was made using the OIC's SPL.

Course Correction

During the *Starting Out* phase of the transition, it became clear that although products could be produced from the product line, the architecture and infrastructure would not be sufficient in the long term. After a technical analysis, a decision was made to implement a new product line architecture.

The new architecture, called Viper, focused on creating composite applications from product line assets. Composite applications are formed by combining functionality drawn from several different sources within a service-oriented architecture and packaging them together into a single-user interface or

work process. In the context of an SPL, a composite application is a member of the product line composed or assembled from the reusable software assets.

Viper implements several aspects of service-oriented architecture, including a high degree of core asset decoupling. The architecture introduced the definition and control of interfaces and asset and data discovery. A common messaging mechanism, or service bus, was introduced to allow software core assets to publish and subscribe to objects, events, and interfaces. A common object model was added to impose object commonality on all software core assets regardless of their origin. A single-user interface was introduced to allow for the assembly of a system that can be presented to the user as a single application instead of numerous applications with different user interfaces. The unified-user interface also enabled an increase in the commonality for functionality such as with editing data (i.e., a single data editor was used instead of multiple data editors). Lastly, the architecture improved product engineering support by providing SPL-attached processes via a Software Development Kit (SDK). The SDK enabled those performing the construction of product line systems to use automation tools for certain aspects of the system assembly process.

The Overwatch Intelligence Center

The first version of the Viper architecture and SDK were completed in 2007 and serves as the core of the OIC SPL. All development relevant to the product line—including legacy code mining, new development, and third-party code integration—uses the Viper architecture. Legacy components are being migrated from the original product line architecture to the Viper architecture.

In its current state, the OIC contains core assets related to intelligence planning, collection, analysis, visualization, and data management. The specific analysis areas addressed by the product line include SIGINT, Human Intelligence, and data fusion (e.g., data correlation, aggregation, and threat estimation). These core assets are designed to be deployed either in the Viper product line architecture or integrated directly into a customer's enterprise architecture.

The product line consists of approximately 270 software core assets with a corresponding number of associated core asset artifacts. The 270 core assets are composed of approximately 900 components and 4.7 million LOC. The origin of the core assets include a legacy code base, newly developed software, and acquired third-party components.

Achievements and Improvements

In spite of missteps at the inception of the effort, there have been several achievements and improvements realized. Two licensed products have been created as members of the product line: an all-source analysis and a SIGINT product. These products are pre-configured collections of core assets that are customized to individual customer specifications.

The all-source analysis product provides data management, link analysis, text extraction, and geospatial capabilities. The first version of this product, based on the original SPL architecture, was developed in less than 90 days. The company estimated that this improvement got products to-market approximately 2.5 times quicker.

The SIGINT product provides collection, analysis, and processing capabilities. The product has been delivered, with modifications and extensions, to two customers thus far. Software reuse between the two customers is approximately 70 percent. There has also been interest from a government customer to acquire these capabilities as an SPL instead of as individual products.

More than 10 customer systems have been completed that are members of the OIC SPL. Software reuse (within these deliveries) is estimated to range between 40 and 70 percent. The software that implements the Viper architecture and SDK have been sold to a U.S. government customer and is serving as the basis for integrating Overwatch Systems, government-developed, and other contractor software. In 2007, the OIC was nominated for inclusion into the SEI's SPL Hall of Fame, which exists to acknowledge excellence in the field and influence in the software engineering community. Metrics to measure other improvements, such as product quality, system cost differential, integration speed, and customer satisfaction are not yet available, but anecdotal improvements have been noted in these areas as well.

Economics

The SEI provides an economic model, called the Structure Intuitive Model for Product Line Economics, that can be used to determine if a product line approach is economically positive for a particular organization [3]. Typically, the benefits outweigh the costs (that is, a positive return on investment) of a product line approach with the third system built from an SPL. Using this as the economic model, Overwatch Systems' experience is in line with the typical return on investment. From 2003 to 2007, Overwatch Systems' revenue nearly tripled. It is believed by management that this growth could not have been achieved without the speed and reduced costs enabled by an SPL approach.

Lessons Learned

During the transition, a number of lessons were learned:

- **Support of organization leadership is critical.** At many points during the transition, if not for the support of senior management—both in terms of funding and organization direction—the effort would not have succeeded. In the case of Overwatch Systems, the general manager, chief architect, and vice president of engineering are product line champions.
- **An architecture specifically designed to support a product line is essential.** Overwatch Systems' first attempt at a product line architecture was rooted in our legacy applications. It failed to properly address certain quality attributes such as modifiability, configurability, and extensibility in the architecture. These attributes were later addressed with the Viper architecture. More infrastructure design and development should have been done before the mining of core assets began. A proper architecture evaluation, as described by the SEI's Architectural Trade-off Analysis Method, is strongly recommended to uncover product line architecture deficiencies [4].
- **It is important to address product line requirements in support tools and processes early in the effort**. Overwatch Systems attempted to change these tools and processes (e.g., requirements management) in parallel with creating core assets. The result was tools that didn't meet requirements, frustration for product line users, and software artifacts that didn't optimally address commonality and variation. Special focus should be made on the requirements for these tools, which are related to creating and maintaining relationships between variation points manipulated in each tool. One way to accomplish this

Coming Events

March 2-5
25th Annual Test and Evaluation National Conference
Atlantic City, NJ
www.ndia.org/meetings/9910

March 2-6
8th International Conference on Aspect-Oriented Software Development
Charlottesville, VA
www.aosd.net/2009/

March 4-5
TechNet Tampa 2009

Tampa, FL
www.afcea.org/events/tampa/09/Introduction.asp

March 22-27
2009 Spring Simulation Multi-Conference

San Diego, CA
www.scs.org/confernc/springsim/springsim09/cfp/springsim09.htm

March 23-26
SEPG 2009 North America
San Jose, CA
www.sei.cmu.edu/sepgna/2009/

April 20-23
21st Annual Systems and Software Technology Conference

Salt Lake City, UT
www.sstc-online.org

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.**

is to determine how the selection of a variation point in requirements is communicated to the related variation point in test procedures.

- **Early in the transition, determine the process for domain analysis that best fits your organization**. Overwatch Systems arrived at a fast, lightweight process, centered on the creation of a high-level architecture artifact that can be performed in the small amount of time that is typically available in our projects. To determine our process, the company examined and used concepts from several references including [5, 6, and 7].
- **Put processes in place to perform domain analysis activities as early in the project life cycle as possible**. Overwatch Systems creates a domain analysis document as early as the proposal stage of a project and it evolves during the project life cycle. This approach ensures that all project participants are in agreement with respect to the project's relationship to the product line.

Conclusion

A product line approach to developing software for government customers is viable and holds tremendous potential for shortening time-to-market and delivering a better value proposition. Adopting a product line approach impacts engineering, technical management, and organizational management aspects of a business and can be adopted by a company like Overwatch Systems in the span of a few years. When executing a transition to a product line approach, special attention should be paid to the product line architecture, the tools and processes that must be modified to support the product line, and techniques related to domain analysis.

In spite of missteps during the transition to a product line approach, Overwatch Systems has successfully produced the OIC SPL. The company has created and delivered multiple software systems from its product line to multiple defense organizations, allowing the government to receive the benefits that a product line approach promises, including decreased development time and reduced costs from planned reuse.◆

References

1. Clements, Paul, and Linda Northrop. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley Professional, 2001.
2. Bosch, Jan. Software Product Lines: Organizational Alternatives. Proc. of the 23rd International Conference on Software Engineering. Toronto, 2001: 91-100.
3. Clements, Paul C., John D. McGregor, and Sholom G. Cohen. "The Structured Intuitive Model for Product Line Economics (SIMPLE)." SEI, Carnegie Mellon University. Technical Report CMU/SEI-2005-TR-003. Feb. 2005 <www.sei.cmu.edu/publications/documents/05.reports/05tr003/05tr003.html>.
4. Clements, Paul, Rick Kazman, and Mark Klein. Evaluating Software Architecture: Methods and Case Studies. Addison-Wesley Professional, 2002.
5. Kang, Kyo C., et al. "Feature-Oriented Domain Analysis Feasibility Study." SEI, Carnegie Mellon University, 1990. Technical Report CMU/SEI-90-TR-021.
6. Cohen, Sholom G., et al. "Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain." SEI, Carnegie Mellon University. Technical Report CMU/SEI-91-TR-28, ESD-91-TR-028. June 1992 <ftp://ftp.sei.cmu.edu/pub/documents/91.reports/pdf/tr28.91.pdf>.
7. Gomaa, Hassan. Designing Software Product Lines With UML. Addison-Wesley Professional, 2004.

About the Author

Paul Jensen, Ph.D., is the chief architect for Overwatch Systems, Tactical Operations. Overwatch Systems makes intelligence planning, collection, analysis, and visualization software for the DoD and other government agencies. He has 14 years of experience in designing and building complex software systems. At Overwatch Systems, Jensen has served as an architect for numerous complex software-intensive projects, led the adoption of an SPL approach to development, and led the adoption of product innovation processes. He has a doctorate in physics from the University of Texas at Austin.

Overwatch Systems
P.O. Box 91269
Austin, TX 78709-1269
Phone: (512) 358-2600
E-mail: paul.jensen@overwatch.textron.com

Software Product Management

Dr. Christof Ebert
Vector Consulting Services

It's easy to confuse the disciplines of project manager and product manager. Simply put, the development of the product or service falls to the project manager, while the market success of software and system products depends on the skills and competence of the product manager. This article provides an overview of software product management and the role of a product manager, and describes concrete practices that can boost an organization's software product management and thus the success rate of products in terms of predictability, quality, and efficiency.

Imagine loggers in a forest. They work hard and cut tree after tree. It is a huge physical effort and their foreman drives them hard to stay on schedule. He wants to cut a certain number of trees per day and provides the workers with all they need to achieve this objective. Suddenly the client shouts, "you cut down the wrong trees!" Despite all the hard work of the foreman and his team, they did not manage to deliver the intended customer expectation. Sound familiar? Indeed, this is what I've observed with many software products. Organizations are pushed to the extreme to be ever more efficient and create products at a low cost, but when it hits the market and sales are lower than expected or customers demand several changes during the development process, margins are dramatically reduced from initial targets.

Successful product management means delivering the right products at the right time for the right markets. Naturally, the success of a product depends on many factors and stakeholders. However, it makes a big difference when a person is empowered to manage a product from inception to market and evolution—and the same person is held accountable for the results. This is the product manager.

At Vector Consulting Services, we have learned from experience with many clients in different industries that success comes from anticipating and meeting the customer's needs together with being on time and on budget. Technical product development—such as for automotive components, communication solutions, defense systems, or IT infrastructure—traditionally focuses on the project perspective and operationally executing a set of given constraints within the triangle of content, budget, and time. Often, it becomes clear too late that customer needs were different from what is built.

Project execution can be rather easily improved by means of CMMI®. Today, there are a lot of exciting results from optimizing projects in terms of cost and cycle time [1, 2]. However, the software product management responsibility and underlying processes remain vague. I often see product definition, road mapping, and marketing decoupled from the engineering project-related processes, which creates deficiencies and overheads such as heavy changes in requirements and missed market opportunities. It is like the loggers: The project runs well, but with the wrong results.

While an organization can embark on the general principles of product management [3], not much specific guidance is available for software product management. This article will provide a small introduction and tutorial on software product management.

What Is Software Product Management?

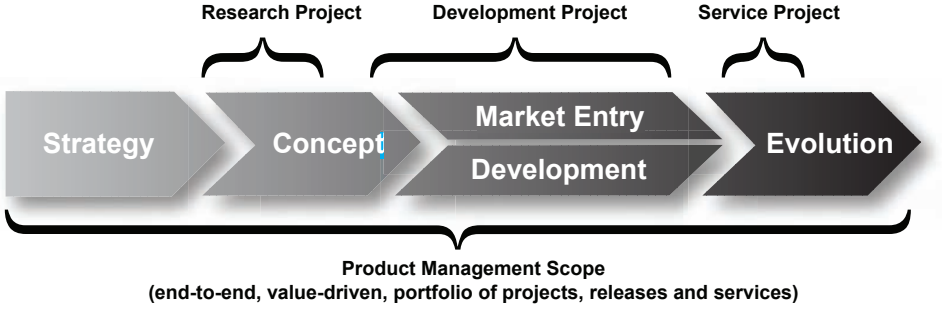
Product management is the discipline and business process governing a product from its inception to the market or customer delivery and service in order to generate the largest possible value to a business. A product is a deliverable that has a value and provides an experience to its users. It can be a combination of systems, solutions, materials, and services delivered. Product management provides leadership to activities such as portfolio management, strategy definition, product marketing, and product development.

Often, the roles of product manager, project manager, and marketing manager are unclear in their distinct responsibilities. To successfully define, engineer, produce, and deliver a product, these three roles need to be clarified [3, 4, 5]. Figure 1 provides an overview of an archetypical product life cycle and shows how different projects integrate towards an end-to-end view of the product. It highlights the differences between managing a project and managing a product. The project is a temporary endeavor undertaken to create a product. The project manager focuses on delivering one specific product or release while meeting time, budget, and quality requirements. The product manager looks to the overall market success and evolution of this product together with its subsequent releases, related services, etc.

To clearly assign responsibilities, there should be three distinct managerial roles:

- The product manager leads and manages one or several products from inception to phase-out in order to maximize business value. They work with marketing, sales, engineering, finance, quality, manufacturing, and installation to make the products a business success [3]. They have business responsibility beyond the single project. They determine what to make and how to

Figure 1: *Software Product Management Spans the Entire Product Life Cycle*



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

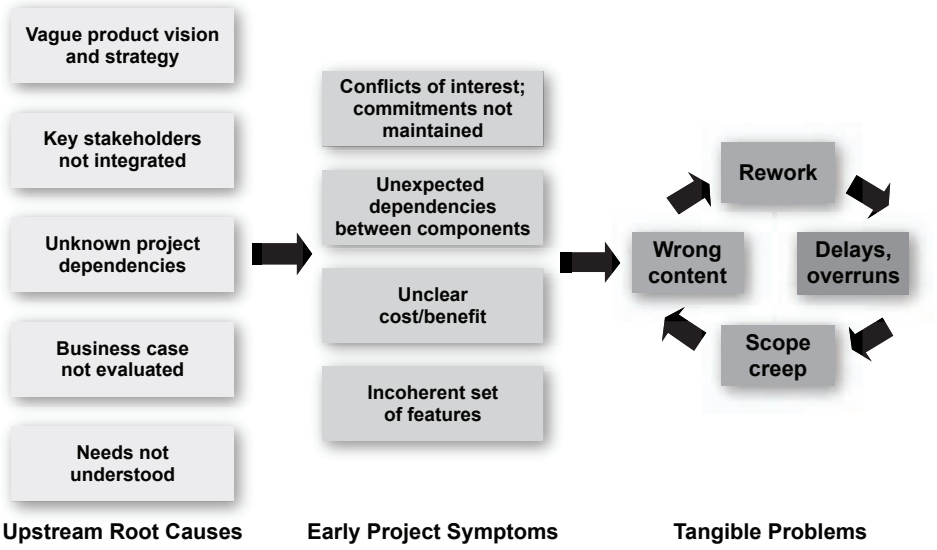


Figure 2: *The Results of Insufficient Product Management*

produce it, and are accountable for business success within an entire portfolio. They approve the roadmap and content and determine what and how to innovate, and are responsible for the entire value chain of a product following the life cycle, asking: What do we keep, what do we evolve, and what do we stop?

- The project manager determines how to best execute a project or contract. They ensure that the specific project is executed as defined and are accountable for business and customer success within a contract project. They manage the project plan and its execution and ask: How do we get all of this accomplished?
- The marketing manager determines how to sell a product or service in order to create a customer experience. They are accountable for market and customer success and have a profound understanding of customer needs, market trends, sales perspectives, and competitors. The marketing manager communicates the value proposition to sales and customers, drives the sales plan and execution, and asks: What markets will we address?

One might argue that in many organizations, one or several of these roles are laid out differently and might simply be coordinating based on directions received from management. While this has certainly been observed, such organizations often encounter interface and responsibility battles and have a lack of ownership as a result. These three roles are necessary and need to be empowered—and held accountable for results. This not only stimulates motivation, but also facilitates faster and more effective decision making in a company [2, 3].

Over the years, Vector Consulting Services has investigated root causes of such insufficient product management and its impacts on hundreds of technical products with different origins, development paces, and sizes [1, 4]. Figure 2 provides an example of how product management failures cause rework, scope creep, and delays. Insufficient product management typically lacks vision, has an unclear market and business understanding, and doesn't involve the right stakeholders (see the left side of Figure 2). This leads to initial symptoms such as a conflict of interest on priorities and contents and incomplete requirements. From here, it's a vicious circle with changes that necessitate rework, which in turn causes delays, which in turn causes scope creep—and so on. Poor product management causes insufficient project planning, continuous changes in the requirements and project scope, configuration problems, and defects. The obvious (yet late) symptoms are more delays and overall customer dissatisfaction due to not keeping commitments or not getting the product they expect (the right side of Figure 2). Being late with a product in its market has immediate and tremendous business impacts [6, 7, 8]. In the contract business, this often means penalties and, in practically all markets, it reduces customer loyalty and overall sales returns.

The tangible problems can't be fixed by pushing a button; instead, the upstream root causes need to be fixed. It would be fatalistic to just take it for granted that requirements changes will always cause delays or that business cases are always wrong. Rather, an empowered product manager acting like an *embedded CEO* (and held accountable for results) will try to fix internal problems and adjust to external constraints and needs—similar to a CEO who cannot simply excuse low performance with bad circumstances. Having worked with different companies in a variety of industries on software product management, we emphasize what we call the *4+1* best practices to optimize product management.

4+1 Product Management Best Practices

Four software product management best practices will improve the situation, if used together. These techniques have been found to reliably improve project performance. A "+1" practice is added to highlight the need for personal competence growth.

1. Install an Effective Core Team

Often, different stakeholders have unaligned agendas that make the project late and cause lots of overhead and rework. The first thing to do is formally create a core team with the product, marketing, project, and operations managers for each product (release) and make them fully accountable for the success of a product. These people represent not only the major internal stakeholders in product or solution development, but also sufficiently represent different external perspectives. The core team leads the product development in all its different dimensions. They typically meet once a week to discuss all open issues, risks, and relevant aspects of the product. Decisions are taken and implemented by the respective function. I suggest announcing and making this core team operational as early as possible in the product life cycle, but certainly when the product or release is defined. The success factor is to give this core team a clear mandate to *own* the project. I have observed that the most need for active support is in the building of an effective core team that agrees that they have to steer the course together. Too often, we face silo organizations in marketing, where product management and engineering don't work together. In many cases, this means the necessity is not only to build teams, but also to train and coach employees and to adjust annual targets

and performance management. As we often realize, culture changes when targets are adjusted.

2. Enforce the Product Life Cycle

Like the core teams, making a standardized product life cycle mandatory for all product releases (i.e., all engineering projects) is essential. Most companies today have such a life cycle defined, but rarely use it as the pivotal tool to derive and implement shared and committed decisions. Too often, requirements changes are agreed on in sales meetings without checking feasibility, and technical decisions are made without considering business case and downstream impacts. A useful product life cycle has to acknowledge that requirements may never be complete and may indeed be in a *continuum* state. The product life cycle should guide with clear criteria (i.e., determining what is good enough or stable enough). This implies that it is sufficiently flexible to handle different types of projects and constraints. This is achieved with basic tailoring techniques and guidance as to which elements are mandatory and which should be adjusted to the specific environment. To foster discipline and visibility, the mandatory elements of gate reviews (such as checklists or minutes) must be explicit and auditable. To reduce overheads, I recommend using online workflow management, which operationally embeds tools and measurements in the product life cycle. Ease of execution with such workflow automation will facilitate reuse, data quality, and consistency. With the current abundance of workflow management systems, I suggest evaluating potential solutions versus your own needs to simplify the process.

3. Evaluate Needs and Requirements

Requirements must be understood and evaluated by the entire core team to ensure that different perspectives are considered. Each single requirement must be justified to support the business case and to allow management of changes and priorities. With our clients, we often found requirements simply being *collected*, yielding lots of unnecessary features that added to complexity—but not to customer-perceived value. In fact, almost half of all delivered features are rarely used and do not provide any payback [1, 7]. If a product is developed on such an unjustified basis, it is in trouble because its requirements will continuously change. A product (release) must address a need and must have a strong business vision. This vision (i.e., what will be different with the release of the product) must be coined into a sellable story. The story then translates to business objectives and major requirements. Good product management first understands the customer's needs and business case, and then develops the necessary features. Requirements are a contract mechanism for the project internally and often for a client externally. They must be documented in a structured and disciplined way, allowing both technical as well as market and business judgment. Their evaluation should specifically look to completeness, consistency, and understandability. Ask a tester to write a test case before processing the requirement. Ask the marketing manager to check whether he or she can sell the feature as described; this avoids unrealistic or overly complex feature lists that don't address real needs. Requirements should not be overly detailed or there is a risk of *paralysis by analysis*; determine what is good enough and ensure that any further insight is adequately considered. After evaluation, requirements are approved by the core team. Only thereafter are the requirements formally allocated to the project, and the engineering effort is spent. Requirements and business objectives must be managed (planned, prioritized, agreed, monitored) throughout the life cycle to assure focus [7, 8]: Have a project plan that is directly linked with the requirements. Work packages within this project plan should show the value they contribute with such links to requirements. Following these directions allows an organization to both focus on what matters and monitor the earned value of the project from beginning to end, as well as proactively manage risks, such as effort being burned without creating value. Also note that your change management needs to be both formal and disciplined, because most issues I've seen in troubled projects result from creeping requirements and insufficient impact analysis. To ease change management, install traceability from requirements upwards to the business case and downwards to test cases.

> *"Too often, requirements changes are agreed on in sales meetings without checking feasibility, and technical decisions are made without considering business case and downstream impacts."*

4. Assure a Dependable Portfolio

Managing release roadmaps—and their own portfolio as a mix of resources, projects, and services—must be the focus of each product manager. Often, roadmaps are not worth the paper they are printed on due to continuous changes that result in a lack of buy-in from sales, operations, and service. Projects are started ad-hoc, while necessary reviews and clean-ups in portfolios rarely happen. With moving targets, sales has no guidance on how to influence clients, and engineering decides on its own which technologies to implement with what resources. The product manager has to show leadership and ensure *dependable* plans and decisions that are effectively executed. Dependable means that agreed milestones, contents, or quality targets are maintained as committed unless a change is agreed on and documented. Be aware that, as a product manager, each ad-hoc content or release change will create the perception that your portfolio is not managed well. Apply adequate risk management techniques to make your portfolio and commitments dependable; as you may find, projects may need more resources, suppliers could deliver late, or technology won't work as expected. For instance, platform components used by several products might use resource buffers, while application development applies the time-boxing technique. If there is a change to committed milestones or contents within the portfolio, it must be approved first by the core team and, where necessary, by respective steering boards and then documented and communicated with rationales.

The "+1": Evolve Your Product Management

Just having these four software management practices distilled and processes agreed upon is not sufficient in order to improve the product management

culture. Often, I've seen organizations where product managers complain about a lack of empowerment and remain in an observer role. The truth is that they simply don't have the right competencies to be empowered as a mini-business owner; this leads to the wrong people in wrong positions. To achieve a true culture change, I strongly recommend competence building for all product managers across an organization. This means change management and closely working with product managers to help them grow. Such individualized and focused competence management strengthens individual product managers and helps them achieve their missions. The equation is simple: Competence and leadership enforced from the bottom up in each project yields better products, which grows motivation and improves the overall performance.

Our software product management framework was shaped by working with hundreds of product managers worldwide in different industries [4, 5]. Figure 3 shows the product management framework in a simplified format. The top shows a product life cycle as most companies today have it formally up and running. Processes are derived from best practices and underline the formal content of product management in an organization. The middle section of the figure shows the typical processes that a product manager is responsible for, or is at least heavily involved with. Finally, what is derived from these processes (shown on the bottom of the figure) are the competency needs of an organization's product management. While there are overlaps across companies, focus areas differ (e.g., a software service provider has different focus areas in this framework than an automotive supplier).

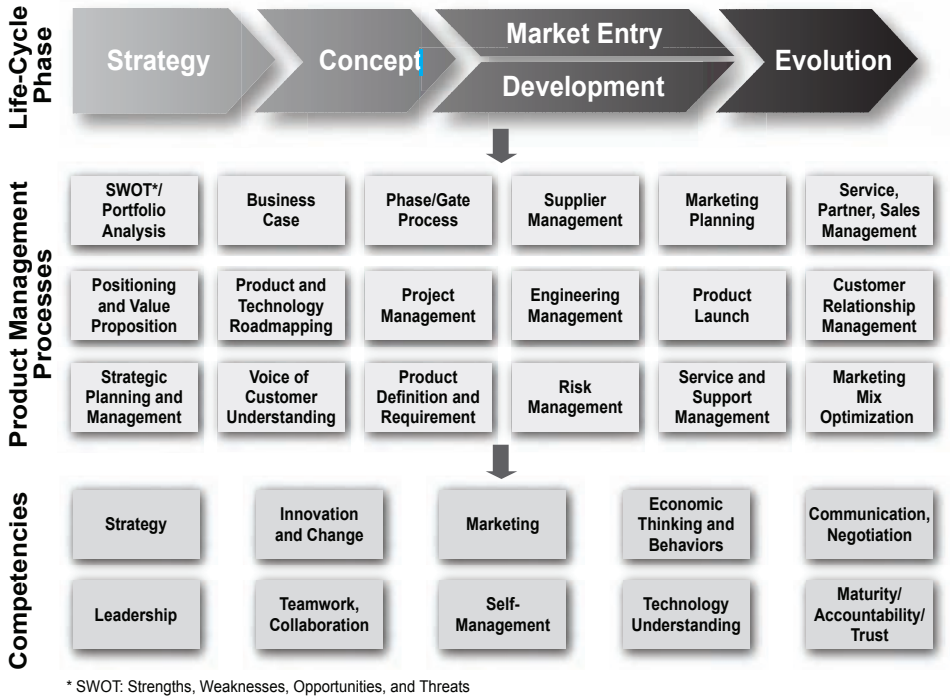
This being done, we can get back to organizational change management and working with each product manager to identify their own strengths and weaknesses. The competencies are used as a basis to provide individualized training and coaching for closing gaps. With good change management and coaching, I've observed a strong motivational push, and have seen (during the competency evolution process) the product management community starting to take shape: Incumbents had a role model (who had been actively trained); an increasing number of product managers became interested in working more methodologically, primarily because they saw the success of other business units and colleagues who had already started implementing the necessary changes [4].

Product managers often ask what they can do to deliver better results. Here are 10 ways to personally grow as a product manager:

1. Behave like an embedded CEO.
2. Drive your strategy and portfolio from market and customer value.
3. Be enthusiastic about your product.
4. Have a profound understanding of your markets, customers, and portfolio.
5. Measure your contribution on sales (top-line) and profits (bottom-line).
6. Periodically check assumptions such as business cases.
7. Take risks and manage them.
8. Foster teamwork based on Lean processes.
9. Insist on discipline and keeping commitments.
10. Be professional in communication, appearance, and behavior.

Having observed hundreds of industry projects from domains such as small software applications and services, embedded systems to large communication and IT systems, I strongly suggest applying the four software product management practices in parallel; they depend on each other. Their combined use will significantly reduce delays and thus improve market performance. These four practices are applicable in different organizations and industries. They are tangible and can be formally introduced to projects during the launch period, thus reducing the impact change and allowing an organization to see the growing benefits early in their projects.

Figure 3: *The Software Product Management Framework*



The Business Value

Does better software product management mean better business performance? At Vector Consulting Services, we have performed a root cause analysis of hundreds of products that underperformed and found similar causes reappearing. Root causes included business cases that were never re-evaluated, unbalanced portfolios that strangulate new products, insufficient management of new releases and service efforts, and a lack of vision causing requirements to continuously change. This is underlined by observations such as in [6], which indicates that the top 20 percent of enterprises deliver 79 percent of new products on-time, while the average enterprise delivers only 51 percent of on-time projects. The same holds true for efficiency: We found that with a requirements change rate beyond 20 percent in a project, productivity falls, and as such, business performance [1].

Improved product management has a profound positive impact on overall business. For instance, strengthening the product management role at Alcatel-Lucent showed that duration (time to market), schedule adherence, and handover quality all improved in a sustain-

able way. We have been working with hundreds of product managers and achieved a 20 percent per year reduction of delays [1, 4]. Explanatory factors for this positive impact of product management include leadership and teamwork, managing risks and uncertainty, mastering stakeholder needs, and accountability towards agreed business objectives—managed by one empowered person across the product life cycle.

Conclusions

Using the 4+1 method means more ownership, leadership, and motivation in product development teams and at their interfaces. Each of the practices can be applied within a single product line if a company is not yet prepared to introduce them across all product lines. The practices and overall product management framework can be gradually introduced to product lines or business units, thus reducing the change impact. Practitioners in engineering, product management, and marketing accept these practices because they yield concrete performance improvement and stimulate empowered project teams.

Growing an organization's product management discipline requires good change management to achieve a culture where these practices are used and implemented by teams across the organization, supported by their management, and communicated openly to resolve conflicts.

For improved software production and market success, product management is here to stay. It is not a proxy to arbitrate a variety of conflicting interests, but rather a key business role in an entire company that is empowered to act as a business owner. It provides the basis for success or failure in the product's development. Or, using our initial analogy: If you do not know which direction to take in cutting the trees, don't simply start just to show progress. Real progress is what creates a lasting user experience, and this is defined from a product perspective—not ad-hoc during project work in a shouting contest.◆

Acknowledgement

This article is based on evolving the product manager competence in different companies worldwide. The 4+1 best practices had been fine-tuned during many discussions at the International Workshop on Software Product Management (IWSPM) series.

References

1. Ebert, Christof, and Reiner Dumke. Software Measurement. New York: Springer, 2007.
2. Reifer, Donald J. "Profiles of Level 5 CMMI Organizations." CrossTalk Jan. 2007.
3. Gorchels, Linda. The Product Manager's Handbook: The Complete Product Management Resource. 3rd ed. New York: McGraw-Hill, 2006.
4. Ebert, Christof. "The Impacts of Software Product Management." The Journal of Systems and Software. Vol. 80, Issue 6: 850-861, June 2007.
5. IWSPM. Proc. from the International Workshops on Software Product Management. 12 Sept. 2006, Minneapolis, and 9 Sept. 2008, Barcelona, Spain.
6. Cooper, Roger G., et al. "Benchmarking Best NPD Practices." Research-Technology Management. Part I: Jan. 2004: 31; Part II: May 2004: 43; Part III: Nov. 2004: 43.
7. Davis, Alan Mark. Just Enough Requirements Management. New York: Dorset House, 2005.
8. Karlsson, Lena, et al. Challenges in Market-Driven Requirements Engineering – An Industrial Interview Study. Proc. of 8th International Workshop on Requirements Engineering: Foundations for Software Quality. Essen, Germany: 37-49. 9-10 Sept. 2002 <www.tts.lth.se/Personal/bjornr/Papers/REFSQ02.pdf>.

About the Author

Christof Ebert, Ph.D., is managing director and partner at Vector Consulting Services. He is helping clients worldwide to improve technical product development and to manage organizational changes. Prior to working at Vector, he held engineering and management positions for more than a decade in telecommunication, IT, and transportation. As a business consultant, author of several books, lecturer at the University of Stuttgart, and public speaker, he has influenced numerous companies with his results-driven contributions.

Vector Consulting Services
Ingersheimer Straße 24
D-70499 Stuttgart
Germany
Phone: +49-711-80670-0
E-mail: christof.ebert@vector-consulting.de

"Spending" Efficiency to Go Faster

Dr. Alistair Cockburn
Humans and Technology

Have you ever been on a project where some person or group is holding up the works? They are called the "bottleneck" station, and here are some usual and unusual strategies for improving output in the presence of various bottlenecks.

In any project or organization, some person, group, or station inevitably acts as a bottleneck to the organization's output. This is, of course, trivially true: Once the output of that station improves so it is not the bottleneck, some other station becomes the limiting factor.

For just that reason, I will not discuss in this article options about improving the performance at individual stations. I will, rather, discuss ways to improve total system results once you have tried all you can think of for the key stations. If you find a way to improve the performance at the bottleneck station, then you get to start all over, working out where the new bottleneck is and how to improve total system performance in the presence of that bottleneck.

Assuming, then, that you have done all you can to improve the output ability at the bottleneck station, it is sometimes possible to further improve output by putting attention on the non-bottleneck stations.

The odd part about these strategies is that when we use the spare capacity at the non-bottleneck stations, we will sometimes deliberately allow "rework" in order to gain an advantage at the bottleneck station. This is counterintuitive to most people: Most of our industry is founded on the notion that we should avoid rework like the plague.

I will refer to this as "spending" efficiency locally for a global gain.

Once you start looking for this, you will see people in ordinary life doing exactly that: Those who have a bit of spare time find ways to help the bottleneck group and streamline the overall flow. The way in which efficiency is best spent differs according to the situation. Taking a look at those alternatives is what this article is about.

A Sketch of the Argument

If you optimize each independent operation in a chain of activities, you are quite likely to not get the best total output from the entire chain. This has been studied and documented for years. It is, among other things, the basis for Eliyahu M. Goldratt's Theory of Constraints [1, 2].

The usual advice is to find the bottleneck station, make it work better until it is no longer the bottleneck, and then pay attention to the new bottleneck station.

While that advice is, of course, correct, it is lacking in two regards:

- For any organization, there is eventually a point when the workers and managers are working at their limit. They may not be able to hire any more people at the bottleneck station or find any other way to improve that group's productivity. This point will be reached in all cases, whether temporarily or due to fundamental limits.

"Those who have a bit of spare time find ways to help the bottleneck group and streamline the overall flow. The way in which efficiency is best spent differs according to the situation."

- There might be other things other people can do to improve the output of the system as a whole.

Here is a simple example taken from ordinary life: John was folding brochures for a part-time job. Sean was doing some other work nearby and wanted to find a way to help, but in this situation couldn't help fold brochures. Sean found that he could help by leaning over from time to time and tilting the stack of papers so John could pull new ones off the top faster.

The point to observe here is that Sean didn't do John's work for him, but still found a way to speed up John's work.

The basis for these sorts of strategies is another truism: People at the non-bottleneck stations have spare capacity. By using that spare capacity in interesting ways, we can sometimes improve the total system output.

Basic Alternatives

When I first went looking for suggestions as to what people should do with their extra time when working at non-bottleneck stations, I found very little.

The first suggestion, given by Goldratt in the context of the Theory of Constraints, is that the people "sit on their hands" so they don't silently turn into bottlenecks without noticing it:

> During a presentation of the five steps of focusing, I can recall Eli saying that at non-bottlenecks people should sit on their hands until there is work to be done. When there is work to be done, then they should work as fast as they can and then return to sitting on their hands. But this act of subordination is extremely difficult as it goes against common practice of maximizing the utilization of every resource, hence the new rules, "Utilization and Activation are not synonymous" and "The level of utilization of a non-bottleneck is not determined by its own potential but by some other constraint in the system." [3]

Goldratt also said:

> ... we have to face the fact that this conclusion means that under no circumstances should we release materials just to supply work to workers ... the worker is not running the machine. He is standing idle. [4]

The Agile and Lean manufacturing communities do one better [5, 6]. They cross-train people at adjacent stations so that if one of them becomes overloaded, the neighbor can do some of his work until the bottleneck moves. These two suggestions—sit idle and do some of the work of the bottleneck station—are good but not always applicable, appropriate, or optimal.

I will develop two more strategies in the following sections:

1. *Simplify* the work of the bottleneck station (as with tilting the stack of papers).
2. Let non-bottleneck people *rework* their ideas to reduce future rework or speed decisions at the bottleneck station.

The second of these is the least obvious and therefore most interesting. I'll illustrate both with case studies from software development projects.

Simplify the Work of Others

The first strategy is based on the company eBucks, a spin-off from a larger bank to create online rewards systems.

The eBucks case has been described in some detail in [7]. Here, I only present details relevant to the current topic: what to do at non-bottleneck stations. I'll break the story into three parts:

1. Organizational structure.
2. Development methodology.
3. Changed strategy.

Organizational Structure

eBucks had about 50 employees at the time, with three programmers who knew the domain and technology well and about 16 programmers fresh from college who were new to both the technology and the business. More such programmers were being hired at the time.

There were four business experts and two expert IT analysts who created and documented new initiatives for the programming team to develop.

Everyone sat within a few dozen steps of each other, in accordance with Agile principles [7]. The company released any new system features they had to the Web every two weeks. The requirements evolved in parallel with development while the programmers were programming.

The company was gaining market dominance in part because they deployed new functionality to the public faster than their competitors could match.

Development Methodology

The four market specialists, in dialogue with their external contacts and with the help of the two experienced IT analysts, would draft function requests (*initiatives*) for the development group to implement.

At the time, there were about 70 initiatives in the queue. Given the proximity of seating and the fact that they were deploying to the Web every other week, the most obvious strategy to consider would be to increase verbal communication between the analysts and the programmers, as is recommended in the standard Agile literature [7].

Closer investigation indicated that this would be a mistake. The problem was that the programmers were overloaded with requests. With four times as many initiatives as programmers, each programmer was working on between four and six initiatives at one time. The programmers were mostly inexperienced, fresh from school, and did not yet understand the problem domain very well. Because of their newness and because of the wide range of assignments in play at any one time, the programmers could not keep the details of their assignments in their heads.

Changed Strategy

The first change, of course, was to reduce the number of initiatives in play, so that each programmer was working on one (or at most two) initiatives in any one week. This was still not enough, given their newness to the domain and the enormous backlog of initiatives they were facing.

> *"The problem was that the programmers were overloaded with requests. With four times as many initiatives as programmers, each programmer was working on between four and six initiatives at one time."*

In the context of this article, the programmers were quite severely the bottleneck station; adding more programmers wouldn't help, nor could the existing ones program faster. The only place found to improve the output of the programmers was to ensure that they were not doing *accidental* rework due to forgetting what the domain rules were, or spending time doing domain research when other people were available to do that. The goal was to keep the programmers programming usefully.

Consequently, we went against the standard advice of the Agile literature and, rather than using verbal communication to pass along the evolving requirements, we agreed that the market specialists and business analysts would write down for the programmers fairly detailed use cases, business rules, and data descriptions.

The analysts then walked the programmers through the text they had written and left the text for the programmers to refer to as they worked.

The result was that the programmers could work in an uninterrupted, *heads down* mode for most of the day, instead of stopping to ask questions or do research.

Lest this seem like the natural, default, or standard solution, it should be reiterated that the reference Agile material on Crystal [7], Extreme Programming [8], and Scrum [9] all recommend reducing written material and increasing verbal material.

In the next project, we did indeed reduce the written material given to the programmers; in that project, however, the programmers were not the bottleneck.

Rework Strategically

The second strategy is based on a medium-sized IT project I call Winifred. The project was a success in the following sense: The team delivered the contracted functionality on time, in three-month increments; the system solved the problem that management was concerned about; the users utilized it as it rolled out each quarter; and the system is still in active use and maintenance 10 years later. Project Winifred has been described in some detail in [10]. Here, I only present details relevant to the current topic: what to do at non-bottleneck stations. I'll break the story into the same three parts as for eBucks.

Organizational Structure

The project was fixed-scope (240 use cases), fixed-price ($15 million), and fixed time (18 months), using several technologies: COBOL, Smalltalk, and relational databases for the production data. It was staffed with 24 programmers in a total team of 45 people (at its peak). It was delivered incrementally to the user base every three months.

The project used technologies in an architecture common in the 1990s: A Smalltalk client on a PC was connected to a server running a relational database that was connected to the company's mainframe system which ran programs written in COBOL. All three technologies—COBOL on the mainframe, relational database on the server, and object-oriented code in Smalltalk on the client's PC—were within the project.

The contractors sat on the same floor and worked closely with the contracting

company's employees. Programmers sat in two or three team rooms; the business analysts had offices several dozen steps away.

The agreement between companies allowed the users to change their minds about what they wanted within each three-month development period. The limit put on this was that within the first six or seven weeks of the 13-week period, the users could change any requirement in any way and could refine their requirements up to week eight or nine. Because testing and deployment preparation work took four to five weeks, there was very little time between when the last requirements changed and when the system was given to the production team.

Two or three Smalltalk programmers were attached to each business analyst in a function team.

The difficulty was that there were only two database analysts (DBAs) for all four function teams. The DBAs quickly became the bottleneck station because they couldn't revise the database fast enough to match the requirements and design changes.

Development Methodology

A detailed description of the methodology can be found in [10]. What is important here are the linkages between the business analysts (BAs) and the programmers, and between the programmers and the DBAs.

The BAs met with user representatives each week to discuss the current release's requirements and to show the progress being made. The BAs documented their meetings with the users only for tracking purposes; they conveyed detailed information to the programmers verbally as needed, daily and after each meeting. Because of the high quality of verbal communication, the BAs were saved from having to revise detailed requirements documents with the frequently changing information.

The programmers attached to each BA changed the domain model as needed from week to week, to keep up with the requirements changes, and also to improve the design.

It soon became clear that the DBAs could not keep up with those changes.

Changed Strategy

The strategies we considered might be enumerated as follows:

1. Wait until requirements settle.
2. Get rid of some programmers.
3. Hire more DBAs.
4. Make the DBAs keep up with the programmers' changes.
5. Have the programmers stop making changes to their design.
6. Let the programmers create trial designs.

The first was not allowed under the terms of the agreement; the second would not speed the project; the third, for reasons of domain knowledge and corporate information security, was not allowed; the fourth was not possible in practice; and the fifth would produce an inferior design. That left the least obvious sixth choice.

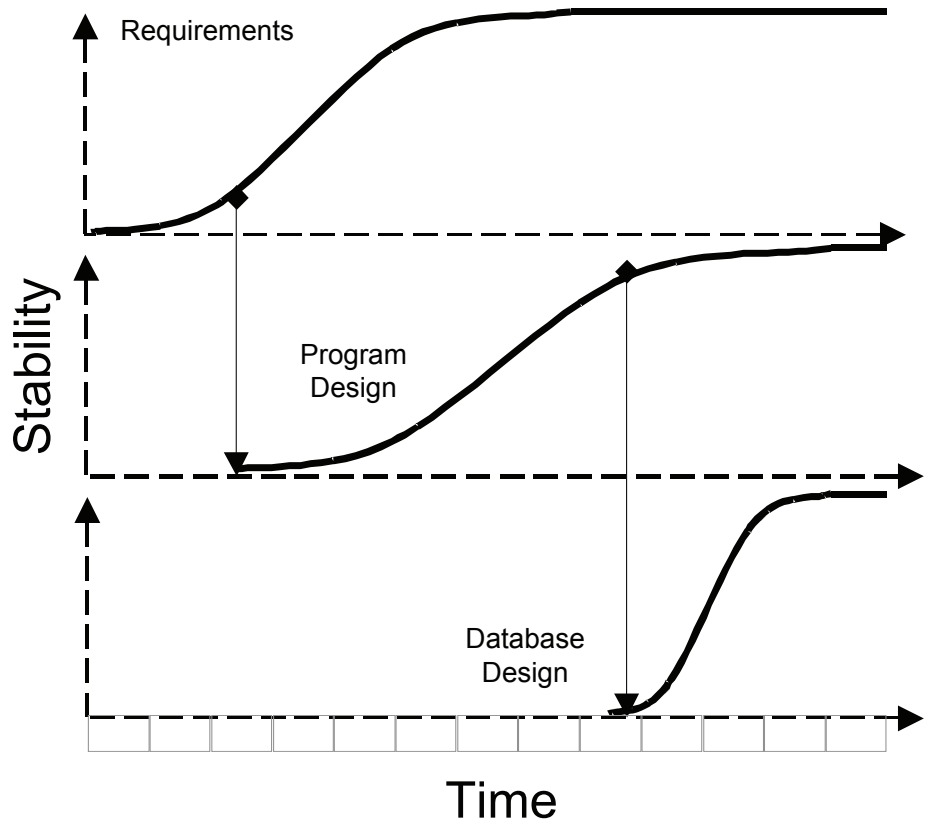
Since there were many more programmers than DBAs (and possibly because Smalltalk is so malleable an environment), the programmers had the capacity to experiment and improve the characteristics of the domain design without impacting the database design schedule. Allowing them to do this could end up speeding the database design work for the simple reason that there would be less design rework later.

Consequently, the programmers did not give the domain model over for database implementation until they had tried several iterations of their design, could assert that it was "relatively stable," and had passed it through a design review with domain experts and the two DBAs.

In terms of what to do with excess capacity at a non-bottleneck station, there is a strategy different from sitting idle, doing the work of the bottleneck, or simplifying the work at the bottleneck; it is to use the excess capacity to rework the ideas to get them more stable so that less rework is needed later at the bottleneck station.

Normally, rework is considered "waste" and minimized, but here we see it used in a strategic manner.

Figure 1: *Project Winifred's Stations Triggered Differently Based on the Stability of the Upstream Material*



Analysis

Four strategies have been named so far for what people might do at a non-bottleneck station:

1. Sit idle.
2. Do the work of the bottleneck station.
3. Simplify the work at the bottleneck station.
4. Rework material to reduce future rework required at the bottleneck station.

In each of these, the efficiency of the non-bottleneck station is lowered, or "spent." This is what I meant in the title, about "spending" efficiency to go faster.

The ways in which the efficiency should be spent is an interesting and meaningful topic for any team, and should be considered deliberately and strategically for the situation at hand.

Rework as a Deliberate Strategy

The rework strategy is the least obvious strategy and deserves some more analysis. To understand its use in Project Winifred, consider the growing stability of the material at the three stations. At the start of each cycle, the requirements were unstable. They became more stable over the first eight weeks of each three-month cycle as they also became more complete.

The domain model was also unstable at the start of each cycle. It became also more stable and complete over time, but only had to reach the top level during the testing period (that is, around week 12 of the cycle). The database design had to reach the point of being stable and complete at the same time as the program. Figure 1 (adapted from [7]) shows how stability grew over time for requirements, program design, and database design.

Recognizing that the programmers had time to rework their designs and that the DBAs really only had one good shot at their design, it was arranged for the programmers to start their work from relatively less complete and stable input, while the DBAs would start their work from relatively more stable input.

The moment of transfer is shown in Figure 1 with the vertical arrow from the upstream station's material down to the downstream station's material. It was important that the people on the respective teams traded information continuously once the transfer had occurred, since the upstream material was changing while the downstream people were working from it.

The strategy of allowing rework is sensitive to the placement of the points of transfer shown in the figure. The fact that the DBAs needed stable information is reflected in the high position of the transfer point: They didn't have time to do rework and therefore they needed stable information. The fact that the programmers did have time to do rework is reflected in the low position of the transfer point and the longer time allocated for program design: They started sooner and reworked more.

If the programmers hadn't had time to revise their domain model, this would have been a poor strategy—the exact point I am trying to make with this analysis.

Downstream vs. Upstream Rework

In Project Winifred, it was the upstream station that had the extra capacity. The strategy we used was for those people to rework their design to improve its quality and to stabilize it before handing it on to the constrained downstream station.

Suppose, however, that it is the downstream station that has the extra capacity. This might happen, for example, when the marketing team is the bottleneck and can't decide which of several alternatives is preferable. Here, the downstream team might use their extra capacity to create several designs for the upstream people to choose from. The unused design would simply get discarded, another example of useful waste.

It is an interesting exercise to imagine the bottleneck station being at different places in the work stream, and working out what a useful rework strategy might be for the non-bottleneck stations.

Summary

Two software development cases and how they made different use of the extra capacity of their non-bottleneck resources were discussed (people whose work was not the speed-limiting factor in the overall output of the organization).

In the first case, eBucks, the non-bottleneck people did extra work to *simplify* the work of the people at the bottleneck stations.

In the second case, Project Winifred, the non-bottleneck people performed strategic rework in order to reduce the later rework of the bottleneck group.

In a thought experiment, we saw how a downstream team might create multiple designs for an upstream group to choose from.

When these strategies are added to the more commonly known ones—of having the non-bottleneck people sit idle or do the work of the bottleneck people—there are five strategies to choose from for how to make use of the "excess efficiency" available at non-bottleneck stations:

1. Have them sit idle.
2. Have them do the work of the bottleneck station.
3. Have them simplify the work at the bottleneck station.
4. Have them rework material to reduce future rework required at the bottleneck station.
5. Have them create multiple alternatives for the bottleneck station to choose from.

For each of these, the efficiency of the non-bottleneck station is strategically lowered; that is, the efficiency is "deliberately spent" in a particular way to gain an overall advantage in system output. The ways in which efficiency should be spent differs according to situation.

The examples in this article were all taken from software development. It should be clear that these ideas apply to organizations and projects in general.◆

References

1. Goldratt, Eliyahu M., and Jeff Cox. The Goal: A Process of Ongoing Improvement. Great Barrington, MA: North River Press, 2004.
2. Goldratt, Eliyahu M. Theory of Constraints. Great Barrington, MA: North River Press, 1999.
3. Bowles, Jim. From a posting on the theory-of-constraints experts mailing list. No URL available.
4. Goldratt, Eliyahu M., and Robert Fox. The Race. Great Barrington, MA: North River Press, 1986.
5. Reinertsen, Donald. Managing the Design Factory. The Free Press, 1997.
6. Personal discussion with Donald Reinertsen, Jan. 2005.
7. Cockburn, Alistair. Agile Software Development: The Cooperative Game. 2nd ed. Addison-Wesley, 2006.
8. Beck, Kent. Extreme Programming Explained: Embrace Change. 2nd ed. Addison-Wesley, 2005.
9. Schwaber, Ken, and Mike Beedle. Agile Software Development With Scrum. Upper Saddle River, NJ: Prentice-Hall, 2002.
10. Cockburn, Alistair. Surviving Object-Oriented Projects. Addison-Wesley Professional, 1998.

About the Author

Alistair Cockburn, Ph.D., is an expert on object-oriented (OO) design, software development methodologies, use cases, and project management. He is the author of "Agile Software Development," "Writing Effective Use Cases," and "Surviving OO Projects," and was one of the authors of the "Agile Development Manifesto." He defined an early Agile methodology for the IBM Consulting Group, served as special advisor to the Central Bank of Norway, and has worked for companies in several countries. More can be found online at <http://alistair.cockburn.us>.

1814 East Fort Douglas CIR
Salt Lake City, UT 84103
Phone: (801) 582-3162
E-mail: acockburn@aol.com

Software Process Improvement Implementation: Avoiding Critical Barriers

Dr. Mahmood Niazi
Keele University

This article seeks to identify perceptions and experiences of practitioners about critical barriers (CBs) that can undermine the implementation of Software Process Improvement (SPI) programs. The objective of this study is to summarize CBs and provide guidelines about how to avoid them. The results of this article provide advice to SPI managers and practitioners on what and how to address CBs when developing SPI implementation initiatives.

There are three major elements involved in SPI initiatives: SPI appraisal, process definition, and process deployment [1]. The SPI appraisal consumes a larger percent of the budget and resources, as it requires money to hire lead appraisers, time away from work for staff to be interviewed, and time away from work for the internal appraisal team. Process defining requires model knowledge, process definition knowledge/skills, and knowledge of the organization/company. Many organizations, however, do not have the model knowledge, the process definition knowledge, or the skills. Often, deployment is not only multi-project, but multi-site and multi-customer type. The whole SPI initiative is a long-term approach and it takes time to fully implement.

A Software Engineering Institute (SEI) report shows the number of months (see Figure 1) required to move from one maturity level of CMMI® to the next [2]. The SPI approach is often considered as an expensive approach for many organizations [3] because, in order to fully implement an SPI initiative, an organization needs to invest enough resources for a long time. This problem is exacerbated if the SPI initiative does not achieve the desired results. Even with the large advances in SPI approaches, the SPI initiatives failure rate is very high (i.e., 70 percent) [4]. This is one of the reasons that many organizations are reluctant to embark on a long path of systematic process improvement.

Thorough literature review revealed that many standards and models exist for SPI, but little attention has been paid to their effective implementation. The chaotic implementation process is the most common cause of SPI implementation failure [5]. Attention to a defined SPI implementation process is essential for the success of any SPI initiative.

This article presents the empirical findings of what can undermine the implementation of SPI initiatives. To focus this study, I investigated the following research questions:

- What barriers can undermine the SPI-implementing initiatives?
- How can one avoid these barriers?

The objective of addressing these research questions is to provide advice to SPI managers and practitioners on what and how to address CBs when developing SPI implementation initiatives.

Research Methodology

This study uses data from interviews with 34 Australian SPI practitioners (15 percent of the requested participants). The target population in this research was those software practitioners who have participated in SPI implementation initiatives. The invitation letter included a brief description of the research project and the nature of the commitment required. In return, I offered to make the research findings available to the participating practitioners.

Software practitioners have cited those barriers that have undermined SPI implementation initiatives within their organizations. Based on their SPI implementation experiences, the practitioners have also suggested guidelines regarding how to avoid SPI implementation barriers. It is worth mentioning that the data was collected from practitioners who were involved in tackling real SPI implementation issues, on a daily basis, in their respective organizations.

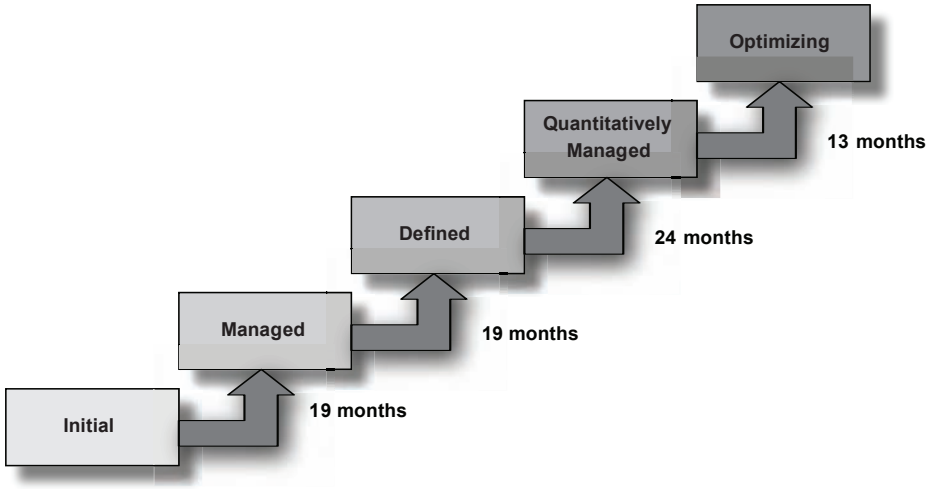
Interviews were conducted with three groups of practitioners:

- The first group was made up of designers/testers/programmers/analysts.
- The second group was made up of team leaders/project managers.
- The third group was made up of senior managers/directors.

All the interview transcripts were read to identify the major themes of CBs. These themes were noted and compared to the notes made during the interviews in order to reassure that the transcripts being analyzed are indeed a true reflection of the discussion in the interviews. This two-step process also verifies that the transcription process has not changed the original data generated in the interviews. Different themes were grouped together under one category. For example, poor response (a user unwilling to be involved, etc.) were grouped together under the CB category *lack of support*. Each category represents a CB for the implementation of SPI initiatives.

In addition to interviews, I have analyzed published experience reports, case studies, and articles in order to identify factors that can play a negative role in the implementation of SPI programs. Each

Figure 1: *Number of Months Required to Move Between CMMI Maturity Levels*



paper was reviewed carefully and a list of barriers was generated.

There were three categories of papers. The first category included papers in which the authors have described their SPI implementation experiences with lessons learned (i.e., why their SPI implementation program was not successful, etc.). It was fairly easy to identify SPI barriers because often authors provided a summary of barriers in the lessons learned. The second category included papers in which SPI implementation was discussed but authors did not provide any summary of barriers. In this case, I have had to read each paper carefully to identify the SPI barriers. The third category included a few papers that I analyzed where the results of empirical studies were described.

In order to reduce researcher bias, I have conducted inter-rater reliability evaluation during this process. Three research papers were selected at random and a colleague, who was not familiar with the issues being discussed, was asked to identify SPI barriers that appeared in the papers. The results were compared with previous results and no great disagreements were found.

For analyzing the data, I used frequency analysis, which is usually the most commonly used approach for similar studies by other researchers [6]. The presentation of data along with their respective frequencies is an effective mechanism for comparing and contrasting within or across groups of variables. In order to analyze the CBs, I recorded the occurrence of a CB in each interview transcript and research article and calculated the relative importance of each barrier.

Findings

Seven CBs were identified that can undermine SPI implementation initiatives: inexperienced staff, lack of defined SPI implementation methodology, lack of SPI awareness, lack of support, lack of resources, organizational politics, and time pressure.

In the following section, these seven CBs are described. For each, guidelines are provided, suggesting how to avoid these CBs.

Inexperienced Staff

In the SPI literature, many authors have described inexperienced staff as a barrier for SPI:

- Kautz and Nielsen describe why implementation of SPI was not successful in one company: " ... the staff and technical director had no prior experience with SPI and its potential benefits" [7].
- Moitra describes the problems and difficulties of managing change for SPI and identifies inexperienced staff as one of the barriers for SPI: "the quality and process improvement people are often quite theoretical—they themselves do not understand quite well the existing software development processes and the context in which they are used" [8].

Software practitioners said in the interviews that the experienced staff should be involved in SPI initiative because they have detailed knowledge of, and first-hand experience with, SPI implementation. With experienced staff, less rework of the documentation items is required and real issues can be resolved. The practitioners said that SPI initiatives can only be successful if staff members have a thorough understanding of the entire SPI process and related business. For inexperienced staff, practitioners emphasized training in SPI skills in order to achieve mastery of its use. This involves equipping the practitioners with the knowledge of the critical technologies (for example, how to measure a process) required for SPI initiatives. The overall objective of this training should be to transfer knowledge to inexperienced staff of SPI activities and interrelated business activities and objectives.

"Seven CBs were identified that can undermine SPI implementation initiatives: inexperienced staff, lack of defined SPI implementation methodology, lack of SPI awareness, lack of support, lack of resources, organizational politics, and time pressure."

The following guidelines were suggested by the practitioners to avoid this barrier:

1. People should be selected for SPI activities who have a track record of different SPI projects.
2. The organization should develop a written training policy for SPI to meet its training needs.
3. Responsibilities should be assigned to each staff member regarding SPI implementation activities (e.g., process design, process testing, and process deployment).
4. A mechanism should be established to monitor the SPI progress of each staff member (e.g., staff members are meeting the deadlines).
5. A mechanism should be established to collect and analyze the feedback data from each staff member and to extract the main lessons learned (e.g., data generated during process testing and results of pilot implementation).

Lack of Defined SPI Implementation Methodology

Practitioners stressed the need to design an implementation methodology that contains an SPI implementation plan as well as SPI activities, practices, responsibilities, and procedures to be used during the implementation process. Often, the SPI projects have no specified requirements, project plan, or schedule [9]. It was recommended by the practitioners to treat SPI as a real project that must be managed like any other project.

Lack of defined SPI implementation methodology has emerged as a CB for successful SPI implementation. This is because little attention has been paid to the creation of an effective SPI implementation methodology. Studies show that 67 percent of SPI managers want guidance on how to implement SPI activities, rather than on what SPI activities to actually implement [10].

The following guidelines were suggested by the practitioners in order to avoid this barrier:

1. SPI implementation methodology should be developed using current technologies (e.g., software tools for planning, tracking, and reporting projects).
2. SPI implementation methodology should be tried and tested in pilot projects.
3. Staff members should be satisfied with the performance of the methodology in the pilot projects.
4. Training should be provided for developing the skills and knowledge needed to successfully use a methodology.
5. Work should be done to continuously improve a methodology with the aim of using it in the whole organization.

Lack of SPI Awareness

Practitioners felt the need for awareness of SPI programs (i.e., return on investment and impact) in order to fully understand the benefits of SPI. Practitioners said that since SPI implementation is the process of adoption of new organizational practices, it is very important to promote SPI awareness activities and share knowledge among different stakeholders. In addition, SPI is an expensive and long-term approach and it takes a long time to realize the real benefits. Hence, in order to get the support of management and practitioners and to successfully continue SPI initiatives, it is extremely important to provide sufficient awareness at the very beginning. SPI implementation is not as beneficial without sufficient awareness of its benefits. With this in mind, practitioners suggested involving all of the staff members in these awareness programs.

The following guidelines were suggested by the practitioners in order to avoid this barrier:

1. The benefits of SPI should be promoted among the staff members of the organization before implementation.
2. Higher management should be aware of the investment required and long-term benefits of the approach before implementation.
3. Staff members should be aware of their roles and responsibilities (e.g., through training and coaching) during the implementation of SPI within their unit of work.
4. Planning should be done to organize and continue SPI awareness events within the organization.
5. Planning should be done to make the SPI a part of the organization's culture.

Lack of Support

Lack of support is one of the barriers that many practitioners think can undermine SPI implementation initiatives. Often, SPI initiatives are not treated as real projects, get low priority, and are easily replaced. As well, management often doesn't support SPI because they do not understand how SPI initiatives can help in their daily work. The practitioners stressed the need to provide sufficient support for SPI initiatives.

The following guidelines were suggested by the practitioners in order to avoid this barrier:

1. Management should show strong leadership and support for SPI.
2. Management should be committed to provide all of the required resources.
3. A procedure should be established to facilitate staff members during implementation.
4. Staff members and higher management should be aware of the benefits of implementation.
5. A mechanism should be established to monitor the SPI progress of each staff member.

Lack of Resources

Management often agrees to SPI without sufficient knowledge of the investment required. In some organizations, management assumes that an SPI initiative will occur with very little investment. In others, management does not consider an SPI initiative as a real project and hesitate to allocate resources.

"In some organizations, management assumes that an SPI initiative will occur with very little investment. In others, management does not consider an SPI initiative as a real project and hesitate to allocate resources."

In addition to the findings from the 34 interviews, the following studies have identified lack of resources as one of the barriers for SPI implementation:

- Florence [11] discusses the lessons learned in unsuccessfully attempting (but not getting) CMM Level 4 at The MITRE Corporation. He states that they achieved CMM Level 3 because sufficient resources were provided, but failed to achieve Level 4 because sufficient resources were not provided.
- Kautz and Nielsen describe why implementation of SPI was not successful because " ... the project managers were hesitant to use resources from their own projects on any improvement activity" [7].
- In the experience of Oerlikon Aerospace, Laporte and Trudel [12] describe five elements for successful implementation of SPI and state that it is important to estimate and provide resources. Otherwise, frustration will end the organization's readiness to adopt the SPI program.

The following guidelines were suggested by the practitioners in order to avoid this barrier:

1. Planning should be done to provide all the required resources (funds, tools, and people) for SPI implementation (e.g., a typical project management activity in which a project manager does cost estimation and allocates required resources for a project).
2. Staff members should be allocated time for SPI efforts.
3. Staff members should agree to the allocated time (i.e., extra time should be allocated for SPI activities).
4. A procedure should be established to avoid *time pressure* (staff members having very little time to complete their tasks).
5. A mechanism should be established so that SPI will not get in the way of day-to-day work (e.g., SPI must be considered as a real project and software practitioners must not be expected to do SPI in addition to their daily software development activities).

Organizational Politics

Many practitioners argued that organizational politics is one of the major barriers in SPI implementation. This is because the SPI is considered a change in the organization and often people resist this change.

Organizations are made up of groups and individuals who have differing values, goals, and interests. The SPI initiative may fit into one group's goals but not into another's. There are many factors that can trigger organizational politics, such as reallocation of resources, promotion opportunities, low trust, time pressures, and role ambiguity.

There are several studies that describe organizational politics as a barrier for SPI implementation. For example, Moitra describes the problems and difficulties of managing change for SPI and identifies organizational politics as one of the barriers for SPI: " ... politics in organizations is probably one of the principal reasons why change management efforts for process improvement initiatives fail" [8]. The writers of [13] conducted a study of 14 companies, investigating some of the important success factors and barriers for SPI; they identified organizational politics as one of the barriers for SPI.

The following guidelines were suggested by the practitioners in order to avoid this barrier:

1. Management and staff members

should provide strong support for SPI.
2. Planning should be done to make the SPI a part of the organization's culture (e.g., awareness training).
3. The benefits of SPI should be promoted among the management and staff members of the organization.
4. All of the key stakeholders should be involved in SPI implementation initiatives.
5. A conflict resolution plan should be established.

Time Pressure

Time pressure is often in the form of meeting project deadlines and getting the product within budget. Practitioners stressed the need to avoid time pressure of staff members during SPI implementation. As discussed in the Lack of Resources section, practitioners suggested that in order to avoid time pressure, SPI must be considered as real work and software practitioners must not be expected to do SPI in addition to their day-to-day software development activities.

There are several studies that describe time pressure as a barrier for SPI implementation. A few of the key studies observed the following:

- In [14], time pressure is identified as one of the obstacles to SPI: " ... operational management feel that in the absence of all other obstacles, lack of time seems to be the overriding obstacle to SPI success in companies."
- Paulish and Carleton [15] describe case studies for SPI measurement and illustrate time restriction as one of the SPI implementation problems.

The following guidelines were suggested by the practitioners in order to avoid this barrier for time pressure:

1. Staff members should be allocated time for SPI efforts and staff members should agree to the allocated time.
2. A procedure should be established to avoid staff from having time pressure (i.e., inadequate time to complete tasks).
3. A mechanism should be established so that SPI will not get in the way of day-to-day work (i.e., SPI should be added to daily activities).
4. The SPI implementation effort should be staffed by people who indicated interest and commitment in the effort.
5. A procedure should be established to facilitate (e.g., to avoid time pressure) staff members during SPI implementation.

Conclusion

The empirical study of CBs with 34 SPI practitioners is presented in this article. Seven CBs that can undermine the SPI implementation effort were identified. The identification of CBs in this study can act as a guide for practitioners when designing SPI implementation initiatives, making it easier to avoid the barriers that have been identified by SPI practitioners who are dealing with these issues on a daily basis. It is suggested that organizations should address these CBs when developing SPI implementation initiatives. This article also provides advice to SPI managers and practitioners on how to address CBs when developing these initiatives.◆

References

1. Garcia, Suzie. Preliminary Insights Working With CMMI in Small Organizations. Proc. of the NDIA CMMI User's Conference, Carnegie Mellon University. Nov. 2003 <www.dtic.mil/ndia/2003CMMI/Garcia.ppt>.
2. SEI. Process Maturity Profile. Pittsburgh: Carnegie Mellon University, 2006.
3. Leung, Hareton K.N. "Slow Change of Information System Development Practice." Software Quality Journal. Nov. 1999: Vol. 8 (3). 197-210.
4. Ngwenyama, Ojelanki K., and Peter A. Nielsen. "Competing Values in Software Process Improvement: An Assumption Analysis of CMM From an Organizational Culture Perspective." IEEE Transactions on Software Engineering 2003: 100-112.
5. Zahran, Sami. Software Process Improvement: Practical Guidelines for Business Success. Addison-Wesley, 1998.
6. Niazi, Mahmood, David Wilson, and Didar Zowghi. "Critical Success Factors for Software Process Improvement: An Empirical Study." Software Process Improvement and Practice Journal 2006: 11 (2) 193-211.
7. Kautz, Karlheinz, and Peter A. Nielsen. Implementing Software Process Improvement: Two Cases of Technology Transfer. Proc. of the 33rd Hawaii Conference on System Sciences, Maui, HI. 2000: Vol. 7, 1-10.
8. Moitra, Deppendra. "Managing Change for SPI Initiatives: A Practical Experience-Based Approach." Software Process Improvement and Practice 1998: 4 (4) 199-207.
9. Stelzer, Dirk and Werner Mellis. "Success Factors of Organizational Change in Software Process Improvement." Software Process Improvement and Practice 1999: 4 (4) 227-250.
10. Herbsleb, James D., and Dennis R. Goldenson. A Systematic Survey of CMM Experience and Results. Proc. of the 18th International Conference on Software Engineering. Berlin, Germany, 1996: 323-330.
11. Florence, Al. "Lessons Learned in Attempting to Achieve Software CMM Level 4." CROSSTALK Aug. 2001: 29-30.
12. Laporte, Claude Y., and Sylvie Trudel. "Addressing the People Issues of Process Improvement Activities at Oerlikon Aerospace." Software Process Improvement and Practice 1998 (4): 187-198.
13. El-Emam, Khaled, Pierfrancesco Fusaro, and Bob Smith. "Success Factors and Barriers for Software Process Improvement. Better Software Practice for Business Benefit: Principles and Experience" IEEE Computer Society 1999: 355-371.
14. Baddoo, Nathan, Tracy Hall, and David Wilson. Implementing a People-Focused SPI Program. Proc. of the 11th European Software Control and Metrics Conference and The Third SCOPE Conference on Software Product Quality. Munich, Germany: 2000.
15. Paulish, Daniel, and Anita D. Carleton. "Case Studies of Software Process Improvement Measurement." IEEE Computer 1994: 27 (9) 50-59.

About the Author

Mahmood Niazi, Ph.D., is a lecturer in the School of Computing and Mathematics at Keele University, United Kingdom. He is an active researcher in the field of software engineering. Niazi has spent more than a decade with leading technology firms and universities as a process analyst, senior systems analyst, project manager, research scientist, and lecturer. He has participated in and managed several software development and research projects. Niazi has a doctorate from the department of information technology, University of Technology Sydney.

School of Computing
and Mathematics
Keele University
Staffs ST5 5BG
United Kingdom
E-mail: mkniazi@cs.keele.ac.uk

Three Encouraging Developments in Software Management©

Esther Derby
Esther Derby Associates, Inc.

When business results aren't what they're supposed to be, companies often make the mistake of trying everything from forced rankings to incentive programs. Instead of these "quick fix" methods, software development managers should consider evidence-based management, as well as Lean and Agile software development techniques, for successful, long-term results.

Twenty-odd years ago, W. Edwards Deming despaired over the state of American management:

> The biggest problem that most any company in the Western world faces is not its competitors, nor the Japanese. The biggest problems are self-inflicted, created right at home by management that are off course. [1]

As I started to write this article, a colleague told me that management in his company is taking decisive action to improve the skill level of employees and increase productivity. This company certainly needs management action to improve business results. They haven't shipped a significant release in over two years, and they've racked up technical debt.

The product development group seems incapable of deciding what they should ship; they change course every few weeks. The result is lots of action, but no completion.

People who are doing their jobs well don't hear from their manager, and neither do those who are failing. Managers complain, but don't clarify expectations, offer feedback, or support people to improve.

The senior managers have decreed that they will raise the overall skill level in the organization and improve productivity. I was intrigued to hear how they planned to accomplish this goal ... until I heard what action they were planning to take. They plan to rank all of the people in the company, with the bottom 5 percent invited to find other employment.

Sometimes, I despair.

But sometimes I dream, because there are moments that make me feel hopeful about management in U.S. companies.

Evidenced-Based Management

Evidence-based management is a movement to look at what actually works in business rather than relying on common practice, fads, and what everyone else happens to be doing.

We've all heard about *known unknowns* and *unknown unknowns*. But Jeffrey Pfeffer and Robert I. Sutton focus on the so-called *knowns* of business practice that *ain't so*, or are only partly so [2].

Pfeffer and Sutton examine the evidence behind the so-called *war for talent*, use of incentive pay, emphasis on strategy, and other widely accepted business practices. In most cases, they find that the data do not support, or only weakly support, the efficacy of these many common business practices.

Take incentive pay, pay-for-performance, and related pay system schemes. Many people in organizations believe that incentives can fix performance problems in organizations. These systems come with cascading goals, objectives, and rating and ranking programs aimed at motivating and rewarding behavior that will benefit the company.

The evidence shows that incentive pay and rating systems overestimate the importance of extrinsic rewards and underestimate the damage incentive pay programs actually do. Pay-for-performance systems not only don't motivate people, they demotivate and often drive behavior the company doesn't want. Yet most companies have some form of incentive pay.

My anecdotal evidence matches the data Pfeffer and Sutton collected. One project manager complained that even though she received a high rating, her annual raise was a pittance—1 percent. On the other hand, a new employee who was still learning the ropes (and was only marginally productive) received a 5 percent raise.

Her manager explained the logic of moving people quickly to the mid-range of the salary base.

While it may sound (sort of) logical, it's the emotional impact that's a problem. "Why should I bust my butt?" the project manager asked. "The best I can hope for is a 1 percent raise. It feels like they don't want me around anymore." Within six months, she'd left the company for a different employer (and a better paying job).

Evidence-based management isn't a quick fix, so I don't expect to see American management transformed overnight. It is a mindset and includes learning and researching, experimenting, and acting on evidence—disciplines that are often at odds with the image of the charismatic, action-oriented executive. As Pfeffer and Sutton say, "Being a 'master of the obvious' may not sound exciting and won't get you labeled as a genius, but it can make and save your company a lot of money" [2].

As a manager, if you're not a master of the obvious, the rest won't much matter.

Evidenced-based management is out there, and many people are reading Pfeffer and Sutton's book or following Sutton's blog. People are starting to ask questions and take management action on the basis of what is known—rather than what they wish was known. I plant the seeds of evidence-based management action wherever and whenever I can, prompting observation, probing for clear thinking, and shoring up hope with facts.

Lean for Software Development

Deming contributed to the post-World War II transformation of Japanese manufacturing. Now, in a sweet turnabout, Americans are examining one of the great success stories of Japanese manufacturing for lessons we can apply in managing software development.

Lean applies key principles that have worked in manufacturing to software development. The Toyota system, from which most Lean thinking derives, has 14 principles. All of them work synergistically, with five that are particularly heartening:

1. **See the system as a whole.** While functional organizations tend to focus on specialized skills and excellence at the component level, Lean teaches managers to see their organizations as systems of interdependent parts. A system view leads to improving the entire system.
2. **Level out the workflow.** Leveling out the workflow involves three things. The first job of management is to reduce overburden. In manufacturing, overburdened machines break down. In knowledge work, overburdened people make

mistakes, fall ill, burn out. The second job is to eliminate unevenness in the workload by figuring out how to create a steady flow. Finally, managers need to eliminate waste. Anything that does not directly add value to a product is considered waste. Table 1 (from [3]) shows the correspondence between the seven manufacturing wastes and the seven software development wastes. Eliminating waste also results in streamlining approval processes, finding the lightest audit requirements that will work, and reorienting budget processes to focus on creating value rather than containing costs.

3. **Create a culture that supports learning and continuous improvement.** The only way organizations improve their results is through feedback, reflection, and problem-solving. One way to support this is retrospectives (a subject near and dear to my heart, and described in [4]). Retrospectives provide a structured way for teams and work groups to examine their technical practices, processes, and collaboration, look for root causes, and plan for improvements. Retrospectives are a plan-do-check-act cycle.
4. **Develop exceptional people and teams.** Investing in people and empowering them leads to better decisions and problem-solving. That means giving people the contextual understanding, skills, and authority to make decisions as close to the work as possible.
5. **Focus on the long-term.** U.S. managers are driven by short-term metrics and financial targets, even when reaching those targets sacrifices long-term results. But to produce exceptional results, managers need to think farther out than the next quarter and cannot cut corners now that will cost the company later.

There is much more to the management philosophy that supports the Toyota Production System[1] and other Lean transformations, but these strike me as the five most critical steps to improving management.

Agile Software Development

Agile methods reinforce key elements of Lean. They aim to manage workflow so that teams work at a sustainable pace—one that they can maintain without burning out. Agile also eliminates waste by focusing only on the most important features, catching defects before they escape an iteration, and reducing documentation to the minimum (and, consequently) allowing the cross-functional development team do their work.

Agile removes the manager from day-to-day task management, and leverages self-organizing teams to create valuable software. The teams manage their own work, and improve their own practices and processes through regular retrospectives. And with the managers freed-up from task supervision, they can turn their talents and energy to removing blocks and impediments that interfere with the team's ability to deliver working software.

The Seven Wastes of Manufacturing	The Seven Wastes of Software Development
Inventory	Partially Done Work
Extra Processing	Extra Processes
Overproduction	Extra Features
Transportation	Task Switching
Waiting	Waiting
Motion	Motion
Defects	Defects

Table 1: *The Seven Wastes of Manufacturing and Software Development*

In many organizations, team's goals cascade down from management. The team works to meet their goals, which in turn contributes to the manager meeting his or her goals. The team is in service to the manager. Agile breaks this dynamic by focusing the team on building the features that have the best return on investment. Teams don't serve their manager, but work towards creating value for the company.

Agile shifts the manager out of task supervision and into enabling productivity. Managers serve the team (and the company) by creating an environment that will enable the team to do their best work and by working across the organization to eliminate waste and impediments.

Conclusion

Clearly the managers at my colleague's company have not yet discovered evidence-based management and have not mastered the obvious. If they had, they'd know that forced ranking won't solve their problems.

They haven't yet realized that their organization is a system, one that is perfectly designed to produce the results they are achieving. They haven't realized that they are punishing workers for management problems, and that improving the skill level of their developers is fruitless without creating an environment where developers can produce results.

Ah, well.

All we can do is plant the seeds, and work with the people who realize their job is not merely telling others what to do, but creating organizations that allow everyone to do their best. And these movements may help us.

References

1. Walton, Mary. Foreword by W. Edwards Deming. The Deming Management Method. Perigree, 1988: xii.
2. Pfeffer, Jeffrey, and Robert I. Sutton. Hard Facts, Dangerous Half-Truths, and Total Nonsense: Profiting from Evidence-Based Management. Boston: Harvard Business School Press, 2006.
3. Poppendieck, Mary, and Tom Poppendieck. Lean Software Development: An Agile Toolkit. Addison Wesley, 2003.
4. Derby, Esther, and Diana Larsen. Agile Retrospectives: Making Good Teams Great. Pragmatic Programmers. Pragmatic Bookshelf, 2006.

Note

1. For more information on the management philosophy behind the Toyota Production System, see "The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer," by Jeffrey Liker.

About the Author

Esther Derby is well known for her work in helping teams grow to new levels of productivity and coaching technical people who are making the transition to leadership roles. She is one of the founders of the Scrum Alliance and co-author of "Agile Retrospectives: Making Good Teams Great." She has a master's degree in organizational leadership and more than two decades experience in the wonderful world of software.

Esther Derby Associates, Inc.
3620 11th AVE
Minneapolis, MN 35401
Phone: (612) 724-8114
E-mail: derby@estherderby.com

Letters to the Editor

Dear CrossTalk Editor,

I have been an avid reader of CrossTalk for many years, starting when I worked at a telecommunications company, through my time at a DoD contractor, and now at a supplier of Military-Standard-1553 hardware and software. It's one of the few publications that I take the time to read on a regular basis because it has always provided me with ideas and tools for improving the processes and the products that my organizations have produced. Not only do I read CrossTalk regularly, but I have also championed it to my employees, who have joined me as avid readers.

I have benefitted greatly from your articles on processes, such as CMMI, DO-178B, Lean, and Six Sigma. I worked at several CMMI Level 5 companies and, through CrossTalk, learned the value of good processes for producing consistent, high-quality software. Now at a mid-sized commercial company, I needed to adapt those heavyweight quality management systems to a more dynamic environment, and CrossTalk has been there with relevant articles and information.

I cannot say with certainty how much money, time, or effort that CrossTalk has saved my employers throughout the years, but I can say that every one of them has benefitted greatly from the ideas and information published in your magazine. I can think of no finer compliment than to say that if I had to keep only one publication to read each month, CrossTalk would be it. My employees and I look forward to every issue, and its articles are the start of many great discussions. It is definitely my pleasure to share with you the tremendous value that I place on your journal, and hope others who value CrossTalk as much as I do will continue to share their comments.

—Robert Miller
<miller@ddc-web.com>

Dear CrossTalk Editor,

I have been a subscriber to CrossTalk since the days of paper in the early '90s and have been highly impressed with the quality and timeliness of the articles provided.

While most IT journals are geared toward either marketing or academic theory, CrossTalk is one of the very few that provide practical, actionable information that is useful in the real world. I especially like the fact that both individual articles and the entire publication are available as a PDF— very useful for offline reading. The information density stands in stark contrast to the froth that adorns most journals.

A while ago, I was tasked with researching the current state of service-oriented architecture, and the September 2007 issue arrived with the entire issue devoted to the subject. I spent a week reading vendor white papers, but it wasn't until I read that CrossTalk that I felt like I was gaining some traction in understanding the state-of-the-art. I was especially gratified to learn about problems and pitfalls to avoid—very welcome advice.

Keep up the good work and thank you and your staff for all your efforts.

—Brian Spaulding
<brian.spaulding@pb.com>

The Value of Toys

During the holidays (a.k.a. the gift-giving season), I noticed that software engineers and my children have a lot in common. No, I'm not going to pontificate on preadolescent behaviors, eating too many sugary snacks, or the two schools of thought based on "giving" vs. "getting." Oddly enough, I discovered a similarity in the bright, shiny, childlike smiles on both children's and engineer's faces when they receive a new toy or gadget. There's a similar look of wonderment, intrigue, and embedded thoughts of limitless playtime. And there's the anticipation of finding new, undiscovered toy functionality for both, whether it's the pop-out wings on a new *Transformers* action figure, or the thumbnail-sized coupling *transformer* hidden inside a new PDA device.

Being a long-time IT manager, it feels like what Yogi Berra called "déjà vu all over again." In my first IT management job, I was told by my supervisor that the way to keep IT professionals happy was to give them big monitors and lots of computer memory. Later, she revised her statement to include the new toys coming out, like desktop video conferencing and duel processors. These new devices have really changed the agility of the workforce and have impacted the way we live. On the other hand, these toys have also given those of us who are still kids at heart a renewed interest in our daily jobs.

Like me, I'm sure many "family managers" (a.k.a. moms and dads) struggle with the pros and cons of technological toys at home, with the cell phone probably being the most divisive. It's great to keep our kids in contact and safe. But the texting ... some say it's destroying our children's personal interaction and spelling skills while others say it's speeding their processing and cognitive abilities.

It's just as complicated of a problem for IT professionals and managers. Portable computing and communication devices have a significant impact on the workforce. Here at the National Nuclear Security Administration (NNSA), we did a formal review of "Portable Computing and Communication Devices." The evaluation had four areas: risk management, responsible management, agility of the workforce, and work-life balance.

Risk management is a big issue for all of us. As managers, we need to recognize that laptops, BlackBerrys, and PDAs contain data, sometimes just in the form of e-mail messages, other times in the form of documents. Employees are usually careful with their devices, but given the nature of the data, the potential risk is usually too great. Don't forget Murphy's Law: Whatever can go wrong will go wrong, and at the worst possible time, in the worst possible way. There are articles in the newspapers every week about how these devices have been stolen and lost with grave consequences. So the first question is: Is it an acceptable risk (personally and professionally) to have this device and information carried on and off-site?

The second area of concern is responsible management. It's the old "need vs. want" debate that we've all had, either with our kids or employees … my sympathies if you're like me, and have had these conversations with both! Parents of the '80s: Do you remember the pre-dawn frenzied shopping stampedes to get our kids a Cabbage Patch Doll? In the 21st Century, it's electronics, and we've either camped out (or seen the campers) awaiting an iPhone or Wii game console. While we can have the "whatever it takes" attitude when it comes to our kids or personal wants, the workplace is a different story. With budget concerns and, yes, the possibility of the old, "so-and-so has one, so why don't I?" complaints, we should think carefully before purchasing. Is this device cost-effective? Is there a real need for this to get the work done outside of the office or is it just a convenience or a status symbol?

There is no question that portable computing devices have greatly enhanced the agility of the workforce. It has never been easier to work off-site without losing productivity and communications capability.

We have all become experts at multi-tasking. Who hasn't sat in a meeting, pretending to "take notes" while really answering e-mail messages or reviewing a document. It begs the question: How much are we missing, and what risk are we creating, by multi-tasking, and not giving our full attention to the primary task? Let me ask the parents out there: What happens when your kids don't do their requisite chores or their grades slip? The toys, or those electronic devices, are the first things to go. If only the workplace was that easy!

We wouldn't let our kids play with their toys 24/7. For adults, it's the accessibility of personal devices, where, actually, *over*-accessibility becomes an issue. Only about a decade ago, we just didn't answer the phone (thank you, caller ID). Today, there are countless ways to work ... and to be reached: e-mail, pager, text message, and so on. You have to really be creative not to be "in touch," and the believability of "I didn't get the message" has shrunk to near nothing. Everyone needs some downtime, and the proliferation of the electronic devices is severely cutting into it. When are you NOT working when you have a portable device? Surprising as it may seem, the 24/7 worker isn't necessarily the most efficient worker. An August 2007 article in *CIO Magazine* <www.cio.com/article/132551> indicated that when people work long hours over a period of time, they actually become less effective, tend to make more mistakes, and increase the security risks.

As IT managers, we have a responsibility to our employees as well as to our companies to maximize the potential of everyone. But by giving the portable devices to everyone who asks for them, are we creating an unacceptable risk to both the company and to the person? The answers to all of the questions I've posed should drive the assignment of the devices, not just desire to have a new toy.

— Dr. Linda R. Wilbanks
Chief Information Officer,
NNSA – Department of Energy
linda.wilbanks@nnsa.doe.gov

Can You BACKTALK?

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

For more information on how to submit your BACKTALK article, go to <www.stsc.hill.af.mil>.

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

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

CrossTalk thanks the above organizations for providing their support.