CROSSTALK

May/June 2009 The Journal of Defense Software Engineering Vol. 22 No. 4

WANTED

RAPID and RELIABLE DEVELOPMENT

GETTING PROJECTS TO THE FIELD FASTER *WITHOUT* REDUCING QUALITY OR CREATING RISKS

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 2009		**00-05-2009 to 00-06-2009**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Crosstalk, The Journal of Defense Software Engineering. Volume 22, Number 4, May/June 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)**	**36**	

Rapid and Reliable Development

Software Engineering Technology

Open Forum

ON THE COVER
Cover Design by Kent Bingham

Departments

CROSSTALK

OSD (AT&L) *Kristin 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. 18.

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.web master@hill.af.mil>.

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

Needing It "Yesterday"

When asking a customer when they need something done, you've most likely heard the reply, "Yesterday!" Why is this answer so common? Possibly it's because customers typically do not request products until they are needed. I liken this phenomenon to my lunchtime habits. I don't go looking for food until I'm saying out loud, "I'm starving!" Both are an exaggeration, but both cause the consumer to seek out those who can deliver products in a rapid and reliable fashion. I know this firsthand through both demanding software customers—and my own complaints during two-minute waits at the drive-thru window. Sometimes we just cannot get things fast enough.

CrossTalk would like to thank the 309th Software Engineering Maintenance Group for sponsoring this issue.

I Googled the combination of the words "rapid" and "reliable" from this issue's theme to further research this need for fast-paced delivery. In less than a second, my search produced more than 45 million results ranging from patient care recovery to back-up recovery software and from cellular service to cellular genotyping. To compare, searching the words "food, air, water, shelter" took twice as long to produce only a half-million results. What can we deduce from this Internet exercise? Possibly nothing, but at least anecdotally (or maybe facetiously), we can presume that consumers crave fast and dependable products even more than the four basic necessities of life.

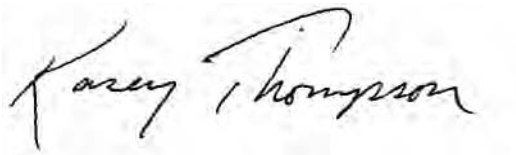
The fact that many of our customers reside in a theater of war only heightens the stakes and drives delivery times to shorter and shorter periods. The shorter delivery time compels the software community to find better methods and tools for increasing the rate of development while maintaining, if not increasing, the dependability of our wares. Just in time, this issue of CrossTalk provides an in-depth examination of such methods.

Ilya Lipkin and Martin Guldahl's article, *Using WYSIWYG GUI Tools With UML* reveals the benefits and hazards of combining these tools to increase productivity and improve rapid prototyping. Timothy J. Trapp, Donald S. Hanline II, Howard D. Kuettner, Jr., and William A. Christian examine software safety concerns when utilizing automated tools for rapid deployment efforts in *Software Safety for Model-Driven Development*. Portia Crowe and Dr. Robert Cloutier—in *Evolutionary Capabilities Developed and Fielded in Nine Months*—show how to quickly and economically use an Agile approach in every phase of a rapid development program.

Also in this issue is *A Distributed Multi-Company Software Project*, where Dr. William R. Nichols, Anita D. Carleton, Watts S. Humphrey, and James W. Over describe how two software development companies were able to jointly develop a large system without reducing quality. Lori Holmes and Roger Heller introduce us to a measurement of previously unaccounted for development project maintenance activities—termed as "impact points," a compliment to traditional function point measures—in *Measuring Maintenance Activities Within Development Projects*.

This issue concludes with Martin Allen's Open Forum article, *From Substandard to Successful Software*. Allen discusses the potential hazards of inferior software products and methods to enhance the probability of creating great ones.

I encourage you to take a break from your hurried environment and enjoy this issue of CrossTalk. Just don't break too long because somewhere today a customer is noticing a software need that you can supply—and they probably needed it yesterday.



Kasey Thompson
Publisher

Using WYSIWYG GUI Tools With UML

Ilya Lipkin
677th Aeronautical Systems Group

Martin Guldahl
309th Software Maintenance Group

This article will discuss the merging of Unified Modeling Language (UML) with "what you see is what you get" (WYSIWYG) graphical user interface (GUI) tools. The topics presented—and discussion of an example with benefits and hazards—will show that the merged solution can increase productivity and provide an improved rapid prototyping platform.

This article is based on the current work done on a project at Hill Air Force Base, which is based on the Team Software Process℠ (TSP℠) practices in the CMMI® Level 5 organization. One of the requirements of our customer is that the project should use a model-driven design UML toolset, namely Rational Rose RealTime. This project is tasked with the design and development of a real-time control system based on C++ auto-generated code. In addition, the project selected to use the Tilcon Interface Development Suite (IDS) in order to rapidly and efficiently create graphical interfaces for the real-time control system that generates stunning displays to the operators of the system at a fraction of the time or expertise otherwise needed.

Overview of UML

The focus of UML is to model systems using object-oriented concepts and methodology. UML consists of a set of model elements that standardize the design description. These elements include a number of fundamental basic model elements and modeling concepts, in addition to views that allow designers to examine a design from different perspectives and diagrams to illustrate the relationships among model elements.

Several views—such as Use Case View, Logical View, Component View, Concurrency View, and Deployment View—create a complete description of the system design. Within each view, an organized set of diagrams and other model elements are visible. Diagrams include use case diagrams, class diagrams, object diagrams, sequence diagrams, collaboration diagrams, state-chart diagrams, activity diagrams, component diagrams, and deployment diagrams. Some key primitive model elements are states, transitions, signals, classes, class roles, attributes, and operations [1].

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

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

The object-oriented principles (OOP) used for the UML design and development are based on the idea of creating self-contained modules that describe desired functionality and interact with others through interfaces in order to create a complete system. To achieve this goal, some of the techniques available with OOP include encapsulation and abstraction [2, 3].

"The UML language is complete enough to allow the creation of auto-generated code that implements the design. The code can be generated from the system description of the model through the use of diagrams and other model elements."

Encapsulation describes the grouping of related functionality, which separates implementation from interface. The implementation details are hidden from outside users, who can only interact with objects of the class through the interface. In this way, the implementation can be more easily changed.

Abstraction provides characteristics of the object or a class that are unique and creates specific defined boundaries with respect to the currently desired solution. Abstraction allows a way of managing system complexity by hiding irrelevant details. For example, it allows for development to continue if a class is just a placeholder for future implementation.

UML Elements for Real-Time Systems Design

Designing real-time systems is a challenge. To address this challenge, an active class model element was introduced in UML. The purpose of this element was to help simplify both the design and the implementation.

The active class model element consists of a communication structure description and a behavioral description. The communication structure is described using a collaboration diagram that shows the ports through which it sends and receives messages to and from other active classes. The behavior is described using a state transition diagram that shows how the active class acts and reacts to its environment[1]. In other words, the active class is a stand-alone capsule of software that talks to its environment through ports (specified in the structure diagram), and performs *actions* as it transitions through a sequence of states (specified by the state diagram).

The characteristics of a run-time system (RTS) object and the UML active class were determined to simplify the process of real-time software design and implementation. In addition, by encapsulating calls to the operating system of the target platform within the RTS, the auto-generated implementation of the UML design can be made largely platform-independent.

The UML language is complete enough to allow the creation of auto-generated code that implements the design. The code can be generated from the system description of the model through the use of diagrams and other model elements [4].

Overview of WYSIWYG GUI Tools

The WYSIWYG concept is a well-known technique that states that the end-product will look, act, and behave the same way as it does being designed on-screen from the developer to the end-customer look and

feel. Currently, there are many tools and products on the market that make the development of the GUIs easier for embedded applications.

One of these tools is the Tilcon IDS. Although there are other tools—such as Altia Design and the Virtual Avionics Prototyping System, which are in many ways similar to Tilcon—the choice over other similar tools was determined using a CMMI Decision Analysis and Resolution (DAR) matrix [5].

The DAR was based on criteria such as customer service and technical support by the vendor, operating system neutrality, cost per use, development seat licenses, performance of the solution, ease of development, and training costs. Each was assigned a weight factor with respect to the importance for the project. In addition, a small prototype was implemented with some of these tools to serve as an input for the DAR. The winning solution was selected from the highest cumulative score.

The UML solution was designed with the idea of neutrality to the GUI development tool. Therefore, if the choice of Tilcon no longer becomes a best-fit selection, the impact to the UML back-end solution will be minimal when going with an alternative GUI.

The Tilcon IDS (see Figure 1) consists of three main components:

1. **The Tilcon Interface Builder.** The Tilcon Interface Builder is a WYSIWYG GUI design tool[2]. An interface is created by *drag-and-drop* of high-level GUI objects like menus, buttons, text boxes, labels, etc. Each of these objects has properties associated with it. These properties, such as color, font, size, meter range, etc., can be tailored to meet a given requirement. As interface development is applied to each object so the application programming interface (API) can manipulate it, the resulting GUI design is saved in a .twd file. The Interface Builder does not require any programming skills to construct a GUI. Non-programmers such as graphic artists can use the Interface Builder to construct a GUI. It is highly recommended that a naming convention be followed so that programmers using the API can access the objects in a consistent way.
2. **The Tilcon Embedded Vector Engine (EVE).** The EVE is a platform-specific engine that renders the graphical interface. It reads the same file as the Interface Builder and ensures that the GUI is exactly the same as seen in the Interface Builder. The EVE runs as a separate process from the application and manages all user events (button presses, mouse clicks, etc.) and handles the screen display. This engine is available for many embedded operating systems, such as VxWorks.
3. **The Tilcon API.** An API is provided to connect the EVE to the application (see Figure 1). The API starts and stops the EVE, facilitates communication between the application and the GUI objects, and allows objects to be created, displayed, modified, or deleted. No low-level, platform-specific graphic calls are required; all of that work is handled by Tilcon [6].

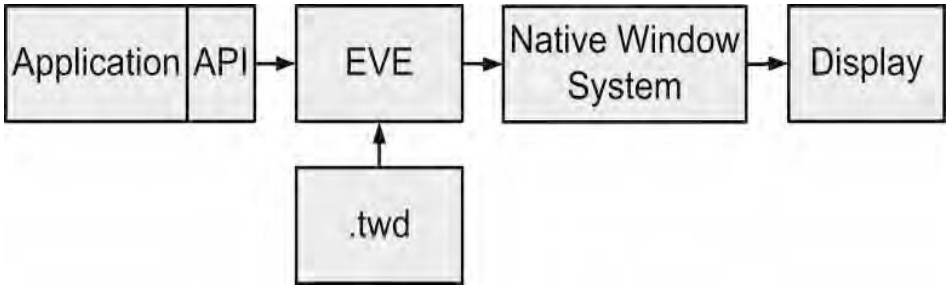


Figure 1: *Tilcon IDS*

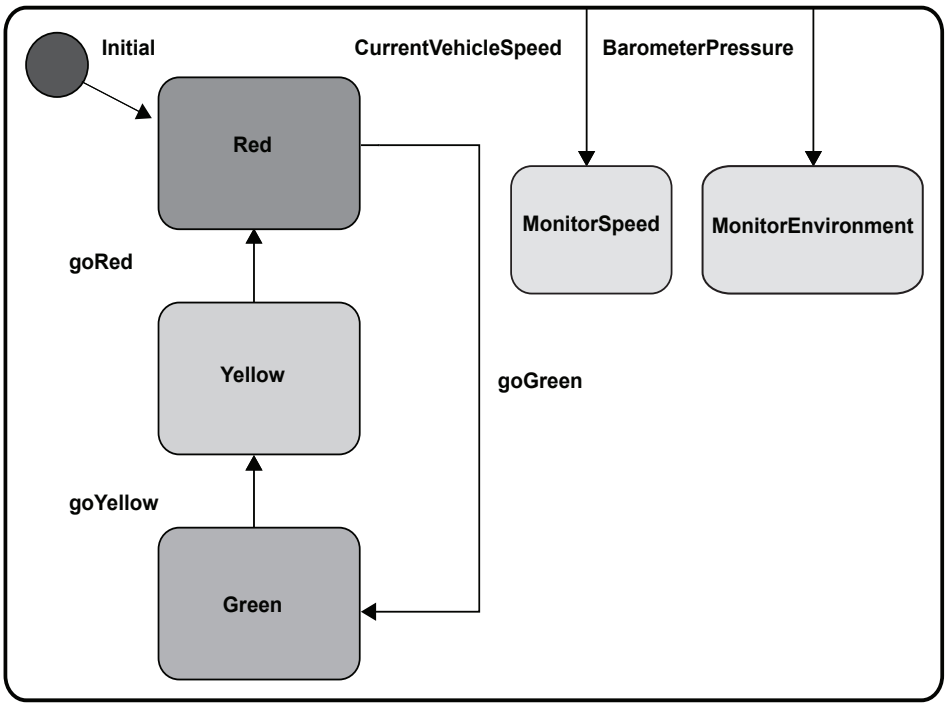
State	Description	Attributes
Initial	System not running	Timer event set to 30 seconds
Green	On timer time out event go to (goYellow) Yellow	Timer event set to 45 seconds
Yellow	On timer time out event go to (goRed) Red	Timer event set to 10 seconds
Red	On timer time out event go to (goGreen) Green	Timer event set to 30 seconds

Table 1: *State Specification Template for the Sequence of Event for the Traffic Light (SEI)* [4]

Merging of UML and a WYSIWYG Interface

Now that the tools have been presented, it is time to discuss the benefits and com-

Figure 2: *Traffic Light Advanced Monitoring System States*



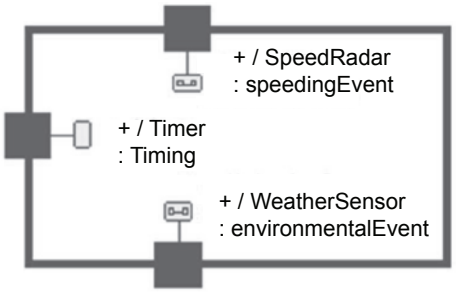


Figure 3: *Traffic Light Input Structure*

mon pitfalls of the approach. To this end, we will demonstrate a simple example using a standard traffic light with some basic gauges for the GUI presentation.

For example, consider Table 1 (on the previous page), which is an SEI State Specification Template [7]. The requirements for the traffic light state that in addition to correctly changing the lights from green to yellow to red and back to green, it will also monitor speed and pressure.

The UML solution shown in Figure 2 (on the previous page) is completely language- and platform-neutral; because the traffic light is drawn in UML, there is no code associated with it (with the exception of event descriptions). Therefore, it is left for the auto-code generation engine to translate UML to a destination language or platform of choice; in this case, C++ [8]. As a result, the amount of source code written is less than 20 lines of code for both the functionality of the traffic light and monitoring systems. The source code written in C++ consists of timer commands in the form "timer.informIn(seconds)" and Tilcon API calls to the graphical objects in the form of "TRT_SetValue(objectID, value)." The Tilcon API calls can be replaced one-to-one with other GUI API calls such as from Altia's "AtSendEvent" if one chooses a different front-end solution.

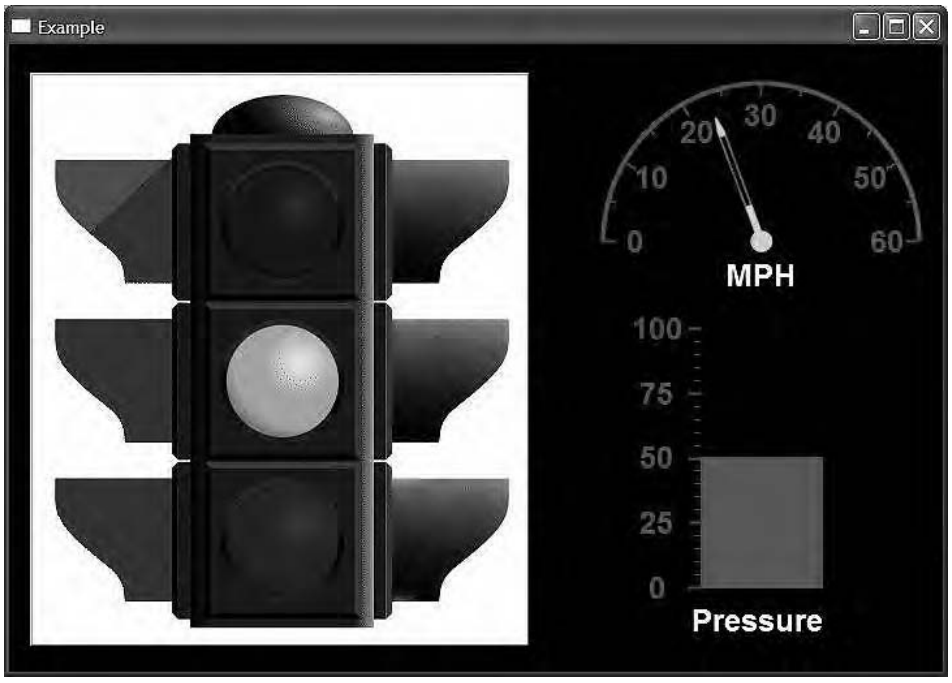
The structure diagram in Figure 3 represents system events that are used to trigger actions on the state diagram. As a result, the stimulus to these events is provided by the system itself in the case of the timer port; as well, external entities such as a radar detector or an environmental sensor feed back data updates for the other ports.

When merging WYSIWYG GUI tools with UML on a complex development project, it is of the utmost importance to exercise good OOP and spend effort to simplify and abstract the WYSIWYG GUI as much as possible from the rest of the UML solution. Abstraction will allow for better unit testing as it is possible to create wrappers that can simulate operator display and input.

The GUI in Figure 4 was quickly created with the Tilcon Interface Builder. This tool supports a drag-and-drop development methodology to create an interface, which consists of several graphical objects listed in Figure 5. The figure lists the object structure with the type and identifier of each object. The entire interface was created and tested without any UML or programming effort or an application back end.

In this example, the description of the mechanics of the speed gauge, pressure gauge, and traffic light animation can be used to demonstrate the effectiveness of this approach at the front end. Therefore, it is of interest to discuss how these objects were created in the Tilcon editor.

Figure 4: *Example GUI*



Traffic Light

The image for the traffic light in Figure 4 (created in Adobe Photoshop) was imported into the Tilcon Interface Builder and placed into a state object. One of many types available in the Interface Builder, this object type can display a different image depending on its state, which can be changed with messages sent to it through the API.

Using Adobe Photoshop with the Tilcon Interface Builder allows for an improved visual experience for the end customer, as graphics generated are generally more visually appealing at a fraction of the cost otherwise incurred if this was done in any other way (e.g., using C/C++).

In order to effectively identify the object for the UML application back end, the use of an API-unique ID such as "StateTrafficLight" needs to be assigned for the screen name. It is important to note: As more complex GUIs with hundreds of graphical objects are created in the Tilcon editor, a strict adherence to a naming convention will be required.

Speed Gauge

The speed gauge is a meter object that was created directly in the Interface Builder. This object is a standard development component that requires a minimum effort of customization. A meter object has many attributes—the range, alarm regions (green and red areas in the scale), tick marks, fonts, and colors—that were all entered in the Interface Builder. For this example, the previously mentioned attributes were slightly adjusted for the visual presentation.

Speed Gauge Needle

The needle for the speed gauge is a needle object that was also created from the standard Interface Builder object type. The needle selected is a predefined object type. Several predefined styles are also available or a custom style can be imported.

Pressure Gauge

Like the speed gauge, the pressure gauge is a standard object available in the Tilcon Interface Builder. It is an object of type "FillMeter" that represents meter position with a fill amount. The modifications for visual effects were primarily the adjustment of the visual width, font color, tick marks, and range.

Speed and Pressure Label

The speed and pressure labels are standard label-type objects. The ID, text, font, font size, and color are all attributes that were entered into the Interface Builder. The primary effort in the case of the labels was to ensure their proper alignment with their respective objects.

Once the graphical objects are entered into the Interface Builder, the GUI functionality can be verified with the Interface Builder operation "Run Test... ." This mode lets the GUI designer verify that the objects operate properly by animating their behavior without the need for a back-end solution; in this case, UML.

What Have We Gained?

The merged solution of a UML and WYSIWYG GUI development tool allows for several advanced flexibilities for the software creation effort. Most of those flexibilities are geared toward rapid development and prototyping. The separation between the front-end WYSIWYG GUI and a back-end UML provides the kind of platform development combination that can bring together technical and non-technical development efforts seamlessly.

Non-Programmers Collaboration

Using a WYSIWYG GUI design tool, it is possible to outsource the generation, prototyping, and user interaction analysis effort to usability experts, graphic artists, and other non-programmers. They no longer need to know anything about UML or any programming language. As well, a GUI object-naming convention should be followed for the project. This will allow programmers to access the objects in a consistent way [9].

Quick Prototyping

With WYSIWYG GUI tools, it is possible to adjust the user interfaces in a matter of minutes—even in the field—rather than hours or days with a comparable native programming solution. It also provides a way to easily evaluate different approaches that otherwise would have taken too long to develop.

Platform Neutrality

The development effort for the GUI is identical regardless of the deployment platform. Whether the target platform is Windows, Linux, VxWorks, or another operating system, the same solution is available and executes identically on the operating systems previously mentioned.

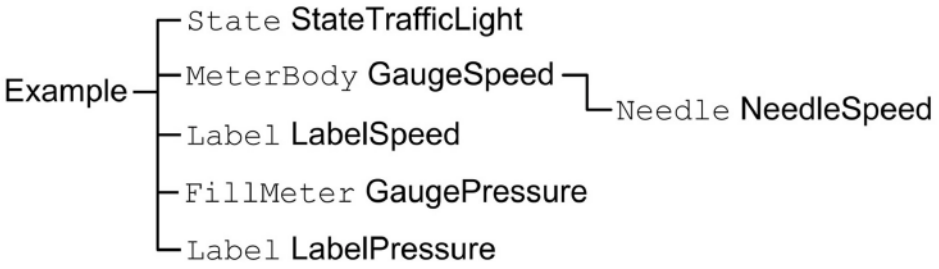


Figure 5: *Example Object Hierarchy*

Therefore, it is possible to deploy the same solution to various other platforms.

Ease of Unit Testing

WYSIWYG GUI development tools allow for automatic editor-based debugging of the GUI designs such that the prototype solution is debugged in terms of visual actions and presentations to the operator. This allows for software developers to concentrate more on the UML part of the solution and spend more time enhancing functionality and quality.

What Have We Risked?

As one might expect, there are always drawbacks to any solution. Over the course of the project, several pitfalls of this merged tools approach have been identified. Here are some of the key issues:

Merging of Design Files

One of the features that has been sorely missed was the ability to merge GUI design files. Because the design resides in a binary file format, the source control tool could not merge them. Therefore, when multiple developers work on the GUI, developers have to be extra vigilant when checking in files to avoid overwriting each other's changes.

WYSIWYG GUI Editing vs. UML Development

There is a fine line between what is considered GUI interface functionality and the UML back-end functionality. Therefore, it is the call of the system architect to identify the separation criteria. When developing in a WYSIWYG GUI tool, it is easy to get carried away with point-and-click, advance animation, and presentation development. While these are great options for some projects, they might not be for others. Some of the functionality that belongs in the UML part of the project should not be moved over to the WYSIWYG GUI development tool.

Let's say someone needs to rapidly change more than 100 objects or text references at the same time on the screen. One approach is to use the WYSIWYG GUI development tool, which results in 100 point-and-click activities; another is to implement the whole thing in the UML back-end for a loop, which can perform the same task in three lines of code. The risk of misplaced functionality can easily wipe out the gains of the merged solution.

Limited Development Language Support

Most of the WYSIWYG GUI tools have a predefined set of software languages that they support. The selection of the WYSIWYG GUI front-end might force the choice of a software language that is not in the best interest of the project, or same language translation must occur. For example, if someone wants to use Visual Basic .NET with a WYSIWYG GUI tool (such as Tilcon or Altia), they will find that it will not be supported and, therefore, be forced to reconsider going with C/C++. The implication of code generated from UML is that it forces a restriction to whatever language the UML tool generates and that this language must be compatible with the API supported by the WYSIWYG tool.

Conclusion

The example presented in this article shows that using UML for the back-end, run-time engine development and a WYSIWYG GUI builder tool for the front-end graphics development can result in overall gains in productivity and ease of prototyping. The event-driven nature of real-time UML facilitates straightforward integration with an event-driven GUI; to some extent, both solutions are platform-neutral. The example demonstrates the ease of these concepts and the integration and simplification of the problem.

One of the key benefits of this approach is that non-programmers can utilize the WYSIWYG GUI design tool to create the GUI. Requirements can be expressed in UML, and these descriptions can be used in the design and implementation of the system. As a result, the development of a complex system is simplified,

in turn minimizing risk, reducing development costs, and shortening schedules.◆

References

1. Sanderfer, Lynn. "How and Why to Use the Unified Modeling Language." CROSSTALK June 2005.
2. Bohn, Christopher A., and John Reisner. "A Gentle Introduction to Object-Oriented Software Principles." CROSSTALK Oct. 2006.
3. Dennis, Alan, Barbara Haley Wixom, and David Tegarden. Systems Analysis and Design With UML Version 2.0: An Object-Oriented Approach. 2nd ed. New York: John Wiley & Sons, Incorporated, 2004.
4. Lipkin, Ilya, and A. Kris Huber. "UML Design and Auto-Generated Code: Issues and Practical Solutions." CROSSTALK Nov. 2005.
5. Chrissis, Mary Beth, Mike Konrad, and Sandy Schrum. CMMI® Guidelines for Process Integration and Product Improvement. 2nd ed. New York: Addison-Wesley Professional, 2006.
6. Tilcon Software Limited. "Tilcon Interface Development Suite White Paper." May 2008 <www.tilcon.com/manual/Tilcon_WhitePaper.pdf>.
7. Humphrey, Watts S. A Discipline for Software Engineering. New York: Addison-Wesley Longman, Limited, 1995.
8. Webb, David R., Ilya Lipkin, and Evgeniy Samurin-Shraer. "Designing in UML With the Team Software Process." CROSSTALK Mar. 2006.
9. Altia, Inc. "How Medtronic Used Altia to Prototype and Deploy Custom User Interfaces for Medical Devices." 12 June 2008 <www.altia.com/downloads/case_studies/Medtronic_Case_Study.pdf>.

Notes

1. In the Rational Rose RealTime tool, active classes are called *capsules*, and the associated collaboration diagrams are called *structure diagrams*.
2. The Tilcon Interface Builder (see <www.tilcon.com/products/interface-development-suite/tilcon-interface-builder> to learn more) is not to be confused with the Interface Builder application for the Apple Mac OS X.

About the Authors

Ilya Lipkin is a project engineer for the 677th AESG/EN Global Hawk Simulations at Wright Patterson AFB. His current research interests include artificial intelligence, human knowledge capture and analysis, neural networks, fuzzy logic, user interface design, software engineering, UML, supply chain control, and customer relations management. Lipkin has a bachelor's degree in computer engineering, an MBA in operations management, and a master's degree in computer engineering. He is currently a doctoral student at the University of Toledo's College of Business Administration.

77th AES Wing
677th AES Group
2300 D ST, BLDG 32
Wright-Patterson AFB, OH
45433-7249
Phone: (419) 290-6017
E-mail: BookWormUT @yahoo.com

Martin Guldahl is an electrical engineer for the Common Aircraft Portable Reprogramming Equipment program, part of the 520th Software Maintenance Squadron, 309th Software Maintenance Group at Hill Air Force Base, Utah. He has more than 15 years of experience in a variety of industry and government positions. Guldahl has a bachelor's degree in electrical engineering and has taken graduate level computer science coursework from the University of Utah. His areas of interest include UML, C++, Verilog, and Perl.

520 SMXS/MXDED
7278 4th ST, BLDG 100
Hill AFB, UT 84056
Phone: (801) 775-4397
E-mail: martin.guldahl@hill.af.mil

Software Safety for Model-Driven Development

Timothy J. Trapp
Raytheon

Donald S. Hanline II
U.S. Army AMCOM

Howard D. Kuettner, Jr., and William A. Christian
APT Research, Inc.

With software applications becoming increasingly complex and the demand for rapid deployment (including rapid prototyping) of software applications increasing, automated tools and updated methods for software development have become necessary. It follows that these new software development processes require new approaches for software safety. One company's 15-element Software Safety Process has now been adapted to a model-driven, spiral software development effort. This process provides an open working relationship to incrementally identify the causes of hazards at different levels.

Software applications are being called upon to perform ever-increasing, safety-critical activities. To that end, software engineering is increasingly using automated tools and updated methods to sustain and gain better intellectual manageability over these solutions. The Missile Defense Agency's (MDA) Global Engagement Manager (GEM) development is an example of integrating these software methodology constraints. At the same time, software safety assurance requirements remain unchanged and mature software safety processes exist. The strategy used in the GEM software development methodology was to leverage the pedigree of existing software safety methods and adapt them to model-driven software development. The advancing software development methodologies lend themselves to a rich, comprehensive approach to safety analysis.

Software Applications Are Becoming Increasingly Complex

The MDA has the mission to provide mechanisms that protect the homeland, deployed forces, friends, and allies from a ballistic missile attack. To frame the problem, ballistic missile defense has a global scale. Threats can originate from any region and be directed at any destination. Since the battlespace will most likely cross multiple areas of responsibility, coordination among command echelons is critical to prioritize the available radar and interceptor resources. To that end, the MDA's Command, Control, Battle Management, and Communications (C2BMC) program has been developing a series of products to enable the DoD to integrate individual sensor and weapon system elements into their Ballistic Missile Defense System (BMDS). One of the C2BMC's products is the GEM.

The GEM's objective is to provide the warfighter with execution-time decision aids to enable them to think globally while acting locally, thereby effectively using the BMDS element resources for the globally integrated active missile defense. Generally speaking, the warfighter's tasks are to:

- Maintain a deep understanding of the active defense design with its branches and sequels.
- Monitor the ballistic missile battlespace.
- Assess gaps from differences in the anticipated and actual enemy courses of action.

> *"The strategy ... was to leverage the pedigree of existing software safety methods and adapt them to model-driven software development."*

- When appropriate, manage by exception[1].

Present warfighter doctrine involves centralized planning while executing in a decentralized fashion. Anticipating the use of automated battle management capabilities to support decisions, warfighters must plan for retaining control over the automation as weapon systems/sensors join and leave the BMDS network. So, the deployment planning process for the BMDS will need to account for effectively integrating technology, processes, and personnel. As the BMD operations are readied for alert, the commander's intent and engagement priorities are configured in the GEM. As suspect tracks are detected and tracked, additional sensor resources can be utilized to gather more information on objects of interest. If tracks are assessed to be a threat, a layered defense based on priority will be calculated. Under an operator's control, weapon-system tasking will be issued to BMDS weapon systems. One can expect responses such as *will comply*, *cannot comply*, etc. This can be due to battlefield effects or conflicts among missions within a multi-mission platform. If an element is unable to support an engagement, the operators can immediately assess and task other elements within the layered defense. Suspect tracks will be prompted to the operators for disposition. If they are promoted to threat status, they will be prioritized and assigned appropriate interceptors. This workflow continues throughout battle.

In addition to being a decision aid, the GEM has a second responsibility, a system of systems (SoS) challenge. In looking at SoS research, failures occur when one system's failure cascades across connected systems or when properly working systems interact in unanticipated ways. The present state of practice is to have a controller manage emergent behavior. Even though fire control remains with the weapon systems engagement function, it is envisioned that the GEM will be assigned the SoS controller role for the BMDS. To globally control effective, execution-speed coordination (in the face of battlefield effects), unanticipated adversary actions, and multiple command structures, the GEM is to be built to have a level of predictability, dependability, and correct behavior that the warfighter can depend on during the *fog of war*.

Software Development Increasingly Uses Automated Tools and Updated Methods

To gain the warfighter's trust in such a mission-critical environment, the GEM decision aid must have predictable behaviors across the broad environmental conditions in which the product may find

itself operating. To achieve that end, the Advanced Battle Management (ABM) development process was crafted from carefully selected modern software development methodologies. The key principles came from model-driven development, model-based acquisition, service orientation, and Agile software development methodologies (as shown in Tables 1 and 2). They were applied to the different activities in the development life cycle (i.e., specification, domain analysis, design, implementation, and testing activities).

The goal is to develop and maintain a common model of the product's behavior so that it optimizes errors and defect exposure at the time they are introduced in the software. A prime contributor to errors and defects are language barriers in systems. This approach strives to create a common understanding across stakeholders by mapping and relating perspectives and points of view into a composite specification. It relates:

- Human-machine interface tasks.
- Functional threads of behavior.
- Use cases to capture desired behavior.
- Collaboration-like activity diagrams, state charts, and algorithm definitions to capture design.
- Code-generated executable models.
- Supporting software to create a dynamic specification that can be assessed.
- Verification cases for further analysis and test.

A second key source of defects is related to the misinterpretation of statically written requirements. This process captures the specification in a way that can be exercised to see if the desired behavior really does occur. This is called an executable model. So, by analyzing this model, one can both evaluate the proper behavior and also reason about the predictable responses of the system when it is faced with faults or conditions beyond its operational bounds.

A complicating factor in software development is identifying errors as they are introduced during implementation. A dominant Agile development methodology tenet used in the ABM development process is short iterations. The design, implementation, and analysis is performed on a small incremental portion of the GEM's required behavior. An iteration consists of completed feature sets with automated test suites. That is, iterations deliver working code with working tests every two weeks. In general, development activities are broken into three major release categories (shown in Figure 1):

1. **Iterations:** Two weeks of development (approximately five iterations per cycle).
2. **Cycles:** Approximately two months of development (approximately six cycles per increment).
3. **Increments:** Approximately one year of development.

Example of Integrating These Software Methodology Constraints

To initially determine and capture the correct dominant behavior, use cases are used as the documentation tool capturing functional behavior from warfighters, analysts, and subject matter experts. Use case extensions are used to reason through behavior under adverse conditions, such as load shedding and fault handling. Safety, performance, mission assurance, human factors, and information assurance concerns are analyzed and incorporated into the use cases. With this behavioral definition captured, detailed design incorporates architectural constraints and further elaborates finer-grained behavior that is only exposed with detailed design.

At GEM's foundation is an architecture that follows well-established business logic. The GEM uses the *kill chain*: detect, track, assign, engage, and assess. Second, from that stable and well-understood structure, behavior that is more apt to change is isolated into components. These components interact only with the stable structure, never with each other. Third, a reactive system of this type demands predictable performance. Between components, the GEM utilizes data queue and blackboard structures—called data stores—for information exchange. In the GEM's case, components must operate concurrently. Each component uses input from prescribed data stores. The safety framework monitors information that flows through the data stores, leveraging policies that are captured in assertions to trigger the safety executive and safety kernel functions into action.

The approach starts with defining policies that capture key behavioral requirements. These policies represent critical behavior that assesses proper operation of the GEM. Many policies are then converted into assertions in logic, linear temporal logic, and/or state charts. By monitoring these assertions in a run-time monitor, fault conditions can be detected, mitigated, or possibly avoided. For safety, critical safety assertions are monitored in the safety exec-

Table 1: *Selected Model-Driven Attributes Used in the ABM Development Process*

Principles from Model-Driven Development	Principles from the Model-Based Acquisition Approach
• Develop a ubiquitous language. • Remove grey-matter translations. • Institute shared understanding of the customer, user, subject matter expert, engineer, software engineer, and tester. • Incorporate model checking and code generation (to allow the specification to be tested without significant investment in implementation).	• Verification of GEM behavior in the specification phase. • Verification of GEM computation code from third-party vendors. • Component-based software engineering. • Verification of modules prior to integration. • Verification of safety components that ensure continuity of operations against all run-time faults. • Separation of computational code from behavior code: ◦ Design by contract. ◦ Verification of assertions in the specification phase. ◦ Asynchronous messaging among software modules. • Test oracles with automated test procedures: ◦ Verified distributed system behavior in the specification phase. ◦ Unified process approach with no more than two-month cycles. ◦ Validation of implementation against the GEM model at the conclusion of each cycle. ◦ Verification of meeting hard real-time deadlines.

Table 2: *Selected Agile and Service Orientation Attributes Used by the ABM Development Process*

Principles From Agile Practices	Principles From Service Orientation
• Short iterations • Automated test • Continuous builds • Re-factoring • Retrospectives • Daily stand-ups • Feature-based development	• **Service reusability:** Logic divided into services with the intention of promoting reuse. • **Service contract:** Services adhere to a communications agreement, as defined collectively by one or more service description documents. • **Service loose coupling:** Services maintain a relationship that minimizes dependencies and only requires they maintain awareness of each other. • **Service abstraction:** Beyond what is described in the service contract, services hide logic from the outside world. • **Service composability:** Collections of services can be coordinated and assembled to form composite services. • **Service autonomy:** Services have control over the logic they encapsulate. • **Service statelessness:** Minimize retaining information specific to an activity. • **Service discoverability:** Services are designed to be outwardly descriptive so they can be found and assessed by discovery mechanisms.

utive. Should the assertion trigger, mitigating action will be taken by the safety kernel. Additionally, assertions are used for mission assurance and at interfaces between components. By establishing assertions about the pre-conditions, post-conditions, and invariants, the errors, defects, and faults can be detected and addressed.

With behavior defined and assertions established, the executable model is created. Code is generated directly from the base logic state charts and activity diagrams. Then, this behavioral logic capability is augmented with the action code. The functionality is layered in during nine-week cycles. It is done in short iterations so all incremental functionality can be evaluated. The proper behavior is analyzed as the executable model functionally grows by monitoring the assertions from an orthogonal view. The test environment is a key analytical tool that is equivalent to an engineer's workbench. Portions of the assertion base are in the test harness and are independent from the implementation. They are always checking to see if a new functionality has broken what had been previously built. It allows for exercising the executable model to analyze and demonstrate anticipated behavior across the broad dynamic range of the battlefield environment. At the end of a cycle, the product leaves the analysis phase and moves into testing. Testing activities that occur at this time are looking for defects that escaped that phase of development.

The ability to economically develop software in this model-driven fashion is made possible by the advancing state of practice in software engineering. These practices continue to make mainstream computer-aided software engineering tools. Though mission assurance and safety concerns are moving into the software development culture in the form of reliability and safety engineering constraints, software safety training is important. Seasoned developers/engineers often have the tools but not the experience and do not know what is sufficiently complete or correct when it comes to these concerns. Some examples of definitions developed for GEM Software Safety training include:

- **Sufficient Completeness.** The consensus of all of the qualified reviewers that the specifications and developments for each part of the presented system and subcomponents are full expressions to the extent that is foreseeable in regards to intended behavior, intended performance, and intended environment.
- **Sufficient Correctness.** The consensus of all qualified reviewers that a software system and its components are free from foreseeable faults in its specification, design, and implementation in regards to intended behavior, intended performance, and intended environment.
- **Intended Behavior.** The planned aggregate of response, reactions, or movements made by a system in any situation. This conversely includes the planned prevention of undesired responses, reactions, or movements.
- **Intended Environment.** Conditions of the elements external to the system that are planned to be affected by or are currently effecting the employment or deployment of the system.
- **Intended Performance.** The metrics of system behavior over time. Examples are latency, throughput, availability, and utilization.

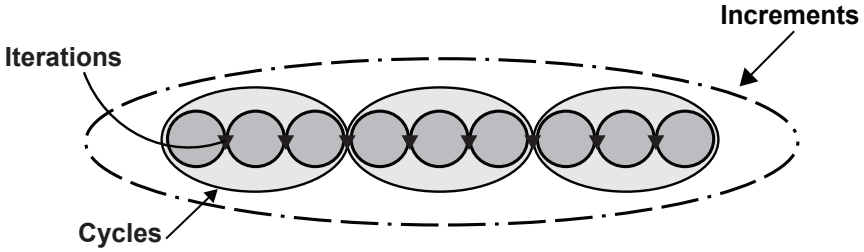


Figure 1: *Three Major Release Categories for Software Functionality*

"Though mission assurance and safety concerns are moving into the software development culture ... software safety training is important."

Requirements and Processes

Safety Assurance Requirements Remain Unchanged

MIL-STD-882D [1] forms part of the basis for the MDA's safety guidance. Awareness of these requirements is the beginning point for software safety training. MIL-STD-882D guidance applicable to software development includes:

- Unacceptable conditions that are considered unacceptable for development efforts. Positive action and verified implementation is required to reduce the mishap risk associated with these situations to a level acceptable to the program manager [1].
- Acceptable conditions are considered acceptable for correcting unacceptable conditions and will require no further analysis once mitigating actions are implemented and verified [1].

A Safety Process for Waterfall Software Development

When software is developed using the classic Waterfall development process, assuring that the software is safe is sometimes difficult. Nevertheless, the software safety process applied to classical software development is understood and practiced by experts today. The 15-element Software Safety Process (developed by APT Research, Inc.), shown in Figure 2 (see next page), is an example of a mature approach.

The fundamental premise is to focus the effort that is needed to perform software system safety. This is classically done by further focusing efforts on the safety subset of the system software. The safety-critical functions are identified from the system requirements documents, the prime development specifications, and the preliminary hazard analysis. Those safety-critical functions with direct or indirect software control are then identified and become the focus for the software system safety effort. The safety-critical software requirements that flow from the safety functional requirements are then identified. The software safety personnel perform this step while coordinating with system safety and software developers.

Adaptation for Model-Driven Software Development

The system-level preliminary hazard analysis provides the framework for reasoning about sub-system (software in this case) hazard analysis in the form of candi-

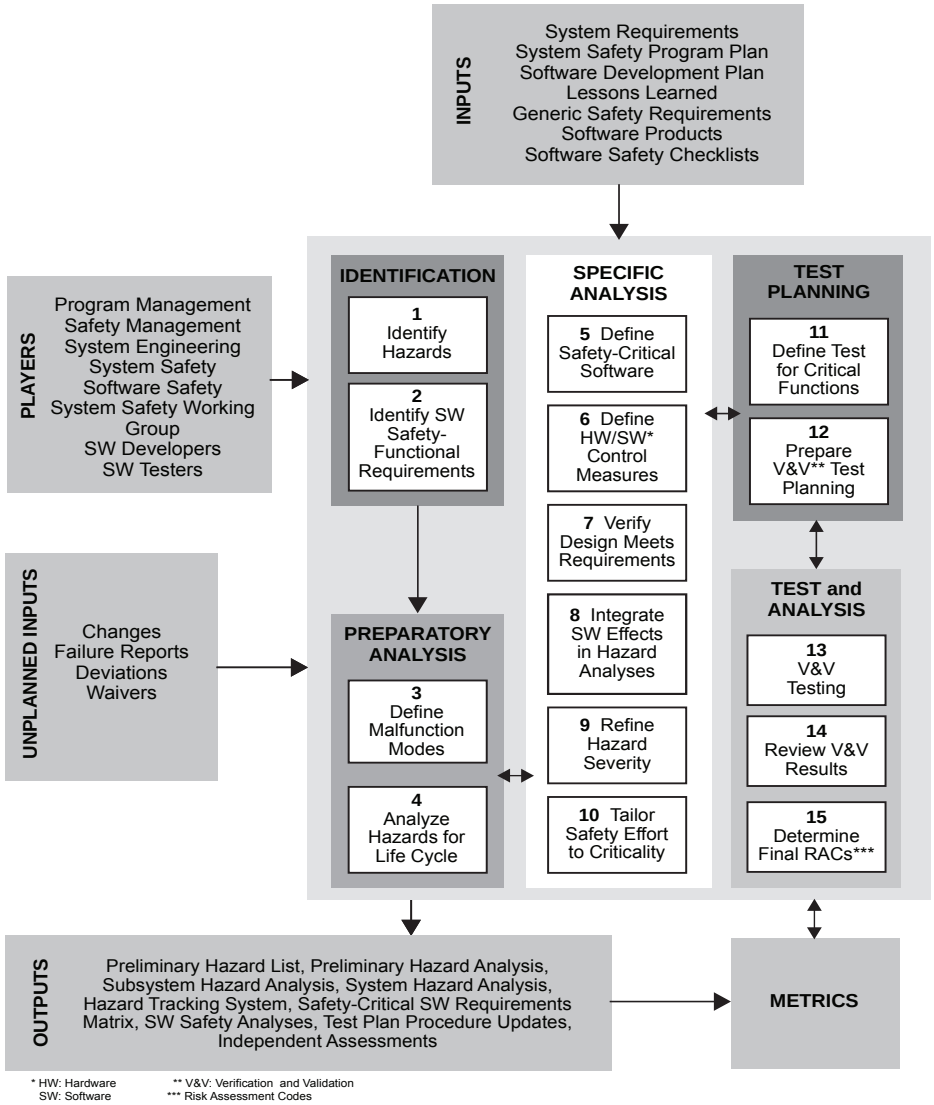


Figure 2: *15-Element Software Safety Process* [2]

GEM Safety Process	15-Element Software Safety Process	Phase
Continuous Throughout Increment	1. Identify Hazards 2. Identify SW Safety-Functional Requirements	Identification
	3. Define Malfunction Modes 4. Analyze Hazards for Life Cycle	Preparatory Analysis
	5. Define Safety-Critical Software	Specific Analysis
	8. Integrate SW Effects in Hazard Analyses 9. Refine Hazard Severity	
	10. Tailor Safety Effort to Criticality	
1.0 Perform Initial Survey of the Capabilities of the Cycle	6. Define HW/SW Control Measures	
2.0 Create and Document Policies for Domain Analysis – Use Cases	6. Define HW/SW Control Measures 7. Verify Design Meets Requirements	
3a. Realization of Controls and Mitigation in Collaborations/Algorithms 3b. Implement Controls and Mitigation	6. Define HW/SW Control Measures 7. Verify Design Meets Requirements	
4.0 Select Verification Evidence Approach for Use-Case Policies	11. Design Test for Critical Functions 12. Prepare V&V Test Planning	Test Planning
5.0 Evaluate Verification Cases for Software Safety	12. Prepare V&V Test Planning	
6.0 Safety Analysis of Verification Results	13. V&V Testing 14. Review V&V Results 15. Determine Final RACs	Test and Analysis
7.0 Create and Maintain the GAR (input to C2BMC Safety Assessment Report)	Updates to Safety Artifacts and Input to C2BMC Safety Assessment Report	N/A

Table 3: *Mapping APT's 15 Elements to ABM Software Safety/Mission Assurance Analysis Activities*

date causes, contributors, controls, and policies within the context of a given required behavior. The differences start here. The analysis activity at the development level is not limited to the safety-critical software functions at first. During each cycle, the incremental specification and implementation draws the whole development staff into identifying broad environmental impacts and captures them in the design artifacts and in a development-level software assessment report called the GEM Assessment Report (GAR). This highlights the value of incorporating the software safety training program into the GEM program. The result is a highly collaborative, fully integrated involvement for:

- Identification of hazards and concerns.
- Evaluation of causes.
- Establishment of detection logic and mitigation policies.
- Selection of mitigator strategies.
- Verification case definition, development, and execution.
- Analysis of results.
- Collection of mitigation assessment and test evidence.

The combination of hazards and concerns coupled with the incremental development activities enhances the defect detection and avoidance mentality. The safety subject matter expert(s) can provide substantial leverage when mentoring the development staff during these activities.

Each of the 15-element Software Safety Process portions was examined for application to each of the development activities. In many cases, the elements were revisited and incrementally built up over the development cycles (see Table 3).

GEM Safety Activities Overview

The GEM Software Safety Activities to support a model-driven, spiral software development effort are shown in Figure 3. The end result is a software safety program that includes all of the fundamental software safety elements, and incrementally grows and matures as the executable model evolves. The beginning of the software development process is the safety entry point into the GEM architecture and product. For the approach to work, it is critical to get buy-in from all stakeholders prior to beginning development.

All members of the GEM development staff should analyze their work in the context of the program's concerns. The following GEM safety principles are integrated into the software development process:

- Reasoning through safety mitigation

options modifies design and implementation trade space.
- Stringent coding standards on safety-critical software are required by MDA safety requirements.
- Analysis is reviewed incrementally by safety staff.
- Safety activities and artifacts are incorporated into the design from the beginning.

Use cases are the mechanisms for specifying GEM safety-required behavior within the GEM behavioral specification. Just as use cases provide the required behavior view, the GEM assessment report provides a view of the key concerns, their causes, contributors, controls, policies, and verifications as they have been defined.

The following stakeholders are to perform safety activities:
- Domain analysts.
- Model designers/architects.
- Model developers/implementers.
- Testers/test analysts.
- Safety subject matter experts.

During the domain analysis, model design, model development, and test, the stakeholders must be alert for the introduction of new hazard causes. New hazard causes may require additional hazard controls and verifications. Additionally, new hazard causes, controls, and verifications must be traced to the implementation and GEM assessment report. In the end, all stakeholders must become familiar with key safety artifacts: hazard logs (or database), hazard causes and controls, and the traceability of required behavior.

System-Level Safety-Critical Functionality Assessment Is Still Fundamental

The fundamental safety premise still holds: Clear enumeration of the agreed-upon safety-critical functions and the assessment of the level of mitigation that exists in the implementation is needed. The safety-critical functions are identified and analyzed in context with the broader environment for which they will operate. Those safety-critical functions with direct or indirect software control are then identified and become the focus for the software system safety effort. The safety-critical software requirements flow from the system-level safety-functional requirements. The software safety personnel perform this step while coordinating with system safety and software developers. However, by using the GEM software safety activities, there is now a rich cause/contributor assessment captured in the development artifacts. Determining the appropriate control to mitigate the causes of the identified hazards has both a top-down and a bottom-up component.

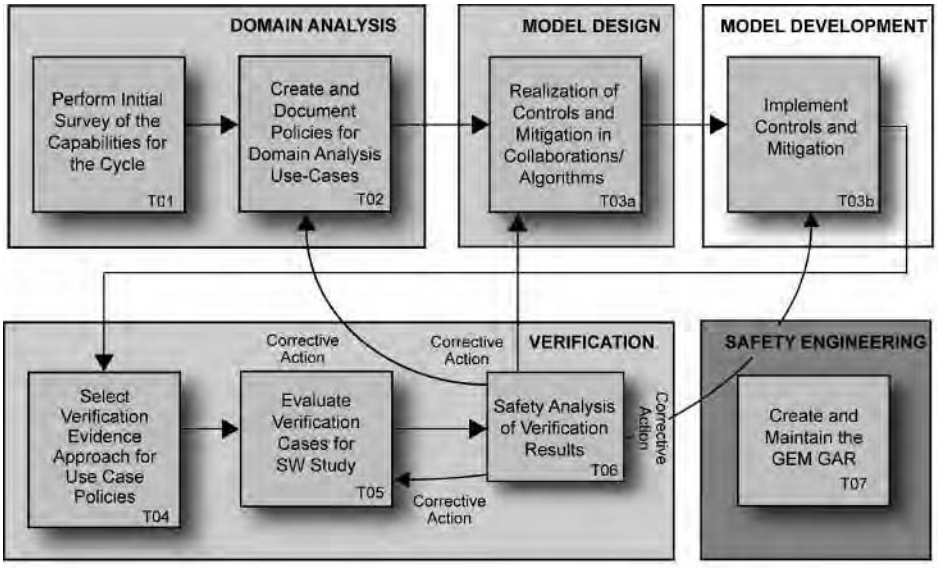


Figure 3: *GEM Software Safety Activities*

Observations and Conclusions

Advancing software development methodologies lend themselves to a rich, comprehensive approach to safety analysis. It provides an open working relationship to incrementally identify the following causes at various tiers of granularity:
- Selection of architecture principles.
- Design and implementation of strategies for mitigators.
- Selection of verification cases that both enhance analysis of properly operating functions and mitigation mechanisms.
- Collection of the needed evidence to satisfy safety review boards.

The GEM software development approach maintains verification logic as additional functionality is added to the growing product base. This software safety analysis complements system-level safety analysis, as is currently being practiced.

In conclusion, a comparison of software safety in Waterfall development projects versus those that use Agile/spiral approaches is found in Table 4.

Additionally, when it comes to modifying the application of software safety analysis for model-driven, spiral-developed software (using the GEM development as an example) this article draws four observations:

1. State of practice in software engineering continues to make mainstream computer-aided engineering tools. Mission assurance and safety concerns are moving into the culture in the form of reliability and safety engineering constraints. Training is important, as seasoned developers/engineers do not know what is sufficiently complete or

Table 4: *Comparison of Waterfall and Agile/Spiral Approaches*

Waterfall Development	Agile/Spiral
Requires complete sets (requirements, high-level design, low-level design, code, and test cases).	Proceeds with partial sets of overall development.
The focus on smaller details is achieved in later phases.	The overall viewpoint is achieved in later phases.
Testing begins later in the development.	Partial sections are tested sooner.
Lessons learned acquired in later phases.	Lessons learned acquired in earlier phases.
The size of the set of changes for correction/ enhancement tends to be larger.	The size of the set of changes for correction/ enhancement tends to be smaller and occur incrementally.
Configuration management is easier since the initial set of requirements tend to be fixed.	Configuration management is harder due to needed response to growing and varying requirements.
It is in later phases that the complete set of details come together that influence safety concerns.	It is in later phases that the overall viewpoint for efficient mitigation implementation can be selected.

correct when it comes to these concerns.

2. System-level preliminary hazard analysis provides a framework for reasoning about software causal analysis in the form of candidate causes, contributors, controls, and policies within the context of a given required behavior. The use of incremental specification and implementation draws the whole development staff into identifying full environmental impacts that are captured both in the design artifacts and a development-level software assessment report. The importance of software safety training cannot be underestimated. The approach enhances the defect detection and avoidance mentality and allows the safety subject matter expert to mentor the development staff and have a high impact during these activities.
3. Safety-critical functions at the system level are used to define safety-critical software functions. These functions are reasoned through at the system level, and use the candidate policies to identify which policies will be tracked as safety-critical. The total approach provides both a top-down and bottom-up assessment.
4. Active safety subject-matter expert involvement is required in software development phase sometimes as the lead, sometimes as a mentor. This allows the program to gain the values afforded by advancing engineering techniques.◆

References

1. DoD. "Standard Practice for System Safety." MIL-STD-882D, Appendix A. 10 Feb. 2000 <http://safetycenter.navy.mil/instructions/osh/milstd882d.pdf>.
2. APT Research, Inc. "The Safety Engineering and Analysis Center." 15 Oct. 2007 <www.apt-research.com/pages/about/S-07-00100_SEAC_Booklet.pdf>.

Note

1. If manage-by-exception actions are warranted, the warfighters are to adaptively direct sensor and weapon system activities in coordination with the element commander. Event-triggered automated actions for elements are coordinated with similar automated actions by the GEM decision aid.

About the Authors

Timothy J. Trapp is the global engagement manager chief engineer with the Missile Defense National Team/C2BMC. He has 25 years experience with the design, development, management, and operations of DoD and commercial interactive systems that are integrated with communications infrastructures. He holds a bachelor's degree in electrical engineering from Purdue University and a master's degree in engineering management from George Washington University. Trapp also holds a patent on the use of multicast-based distribution for timely and real-time data.

Raytheon
2611 Jefferson Davis HWY
STE 700
Arlington, VA 22202
Phone: (703) 418-4288
E-mail: tim.j.trapp@mdnt.com

Howard D. Kuettner, Jr. is the software safety lead for a major system development program at APT. He has more than 35 years experience in systems and software development, systems and software test, and systems and software safety. Kuettner has been a member of the System Safety Society (Tennessee Valley Chapter) since 2000, and was named their Engineer of the Year for 2003. He has a bachelor's degree in physics, and has co-authored papers presented at prior International System Safety Conferences.

APT Research, Inc.
4950 Research DR
Huntsville, AL 35805
Phone: (256) 327-3383
E-mail: hkuettner@apt-research.com

Donald S. Hanline II is a system safety engineer for the U.S. Army Aviation and Missile Command (AMCOM). He has more than 25 years of experience in the aerospace industry, with nine years of systems safety and software systems safety experience on command and control and weapons system development programs. Hanline serves the AMCOM Safety Office on independent safety review boards, the development of Army and AMCOM software system safety requirements, and software system safety training. He has bachelor's degrees in chemistry and mechanical engineering from the University of Alabama in Huntsville.

U.S. Army AMCOM
ATTN: AMSAM-SF-A
Redstone Arsenal, AL 35898-5000
Phone: (256) 842-3248
E-mail: donald.s.hanline@us.army.mil

William A. Christian is currently a software safety engineer for the GEM at APT. He has more than 30 years experience in the hardware and software requirements, design, implementation, and test. Christian has co-authored a paper on reviewing code for requirements verification and has spent more than 20 years in developing software for instrumentation, intercom systems, databases, and testing radio frequency applications.

APT Research, Inc.
4950 Research DR
Huntsville, AL 35805
Phone: (256) 883-3474
E-mail: bchristian@apt-research.com

Evolutionary Capabilities Developed and Fielded in Nine Months

Portia Crowe
U.S. Army, Program Executive Office C3T

Dr. Robert Cloutier
Stevens Institute of Technology

The DoD is facing challenges to rapidly deploy operational capabilities in complex environments where bridging legacy and new technologies are key to success. The challenges arise as a result of diminishing budgets and the need for new capabilities to operate in war environments, including the global war on terrorism. To balance this imperative need with rapid response, we found that our developed Agile life-cycle paradigm was a viable solution to meet challenges brought about by changes in the environment. This article demonstrates how a DoD program used an Agile approach, throughout every phase of the program's life cycle, to rapidly field capabilities.

Our intent with this article is to give DoD programs another successful data point for implementing an evolutionary acquisition strategy using Agile life cycle processes, and share our learned tenets of this process in the hopes of helping others rapidly field capabilities. In an environment where requirements are unforeseen and quickly changing, we need our systems to be flexible and adaptable to meet these growing challenges. Before 2006, the U.S. Army readiness reporting system had no longer met the needs of commanders in providing timely and detailed data to make informed decisions. Complex environments and service-oriented architecture (SOA) changed the landscape of operation and the systems that were operating in it. The challenge was to, within nine months, transform an old Army readiness system to meet current needs without losing existing capabilities while also developing key functions currently needed by commanders.

The solution was for the Product Manager Strategic Battle Command together with the Headquarters–Department of the Army G3/5/7 to modernize the legacy Army readiness application, PC ASORTS, by creating the Defense Readiness Reporting System-Army (DRRS-A). The DRRS-A aligns with SOA strategies and supports the demands of new requirements, capabilities, and modifications in the areas of force registration, force readiness, force projection, and mobilization. The DRRS-A team included about 60 people from the government and multiple contracting teams. The strategy was a phased approach allowing for the deployment of high-priority capabilities first and then subsequent capabilities using an incremental process. The DRRS-A software system first deployed in late 2006 after only nine months of development and has fielded new capabilities incrementally in as soon as two months [1]. The program consists of secure Web-based capabilities such as unit status reporting that details mission-critical information including personnel levels, training status, equipment availability, and equipment serviceability. It is used as a commander's assessment tool as it reports a unit's capability to execute missions. Using an evolutionary strategy, the legacy application was a stepping stone for the development of new capabilities and rapid, yet disciplined, transition of processes.

Approach

The DoD embraces change after a long history of Waterfall software methods and single-step to full-capability approaches. The goal of the DoD's Evolutionary Acquisition (EA) policy is to provide operational capabilities to the warfighter—quicker than traditional methods—through rapid incremental fielding, building to full-objective capability [2]. The DRRS-A implementation plan included using the DoD EA approach as a guideline for delivering capabilities in increments. Our evolutionary strategy was to take the existing readiness system and perform system modernization in a phased approach, which included leveraging functionality inherent in the old system and translating it into usable functionality in an SOA, combined with serial guidance and directives issued by the Joint Chief of Staff. The DRRS-A currently has as many as 5,000 users including Army Commands, the National Guard Bureau, Army Forces Command, and the United States Army Reserve Command (USARC).

> *"In an environment where requirements are unforeseen and quickly changing, we need our systems to be flexible and adaptable."*

Our system methodology was to integrate all aspects of program life-cycle phases using an Agile approach with rapid prototyping to ensure that the customer and user needs were met. We took a linear life-cycle approach and worked life-cycle phases in parallel and often at the same time. Working within an aggressive schedule, we carried out continuous facilitation of the following phases:

- Concept refinement, requirements, and architecture analysis and design.
- Capability and software development.
- Integration, testing, and demonstrations.
- Production and deployment.
- Operations, support, and training.

The user community consistently worked with the developers to refine concepts and requirements to be developed. In a month, the team could get as many as five new requirements and enhancement requests from various sources, such as the readiness community and the Joint Chiefs of Staff. New requirements can range anywhere from new calculations to new information required from the user. During testing time, the team mainly focuses on fixes and performance of the applications. The rapid and iterative software development process included conducting continuous integration and testing on a daily basis by checking in and out software code. Scrum, our Agile process, was implemented in 30-day software development sprints using a prioritized requirements list also known as a *backlog*. It helped keep the focus on user needs with demonstrations at the end of each interval [3]. In a month, the development and test teams can work through as many as 15 requirements. The Scrum development process [4] is shown in Figure 1 (see next page).

Characteristic	Comments
Liberty to be dynamic	Agility needs dynamic processes while adhering to acquisition milestones.
Non-linear; cyclical and non-sequential	The life-cycle behavior was not like traditional waterfall models or linear frameworks; decreasing cycle times.
Adaptive	Conform to changes, such as capability and environment.
Simultaneous development of phase components	Rapid fielding time may not lend to traditional phase containment (i.e., training and software development together).
Ease of change	Culture shift to support change neutrality; ease of modification built into architecture and design.
Short iterations	Prototyping, demonstrating, and testing can be done in short iterative cycles with a tight user feedback loop.
Lightweight phase attributes	Heavy process reduction, such as milestone reviews, demonstrations, and risk management.

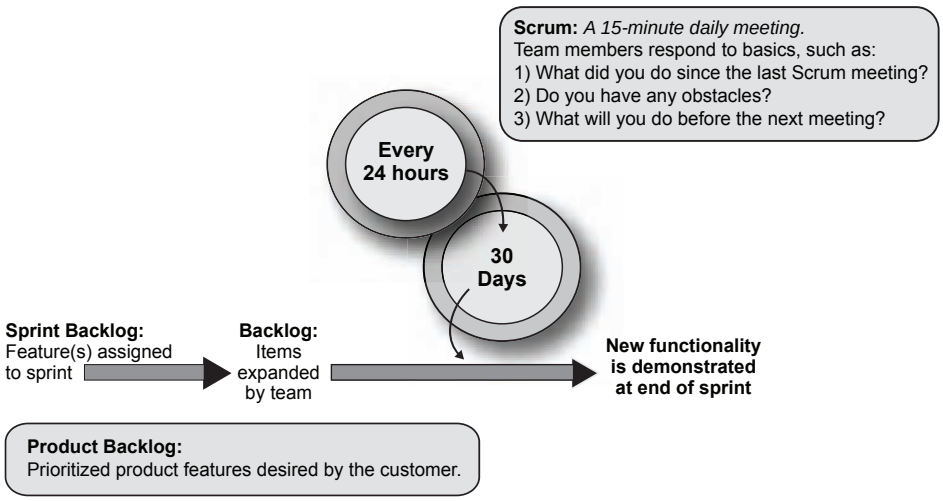
Table 1: *Emergent Agile Characteristics for Rapid Prototype and Development*

At the end of each development sprint (estimated as every 30 days), a training team of six people would develop or edit training materials, user guides, and train the help desk on these features. The user community is trained on the new or edited features via remote, computer-based, and/or face-to-face training. The imposed user feedback loop gave us greater confidence that we were building to expectations and user requirements. We were inline with priorities as a result of canvassing for feedback during bi-weekly iteration meetings, testing events, and surveys [5].

The key to an Agile life cycle is to keep everyone informed so that functions can be done in parallel. To ensure continuous coordination, all of the functional leads (i.e., system engineers, development, logistics, and requirement proponents) and key users were at every sprint review. This helped to keep the team in sync on new application features, training needs, integration and test events, and priorities of requirements.

Interoperability and integration testing was frequently conducted not only at the subsystems level but for system of systems and external dependencies. Prior to major releases, we found three to be the *magic number* of user dress rehearsals or test events. These allowed users to test functions of the applications and engineers to get a good read on the performance of the system. Participants (usually 20 to 30 users) were chosen by the functions or applications being tested during that event and the location of participants (such as Iraq or Afghanistan), which gave us good performance data. These testing events, each lasting three days, usually started three months prior to a big release and were about three to four weeks apart. Coordination with participants, training, and test procedures distribution occurred prior to each event. An online survey was prepared for the users to track their issues and concerns during the event. We recently added performance questions to track how fast the functions were loaded and displayed. After each day of the test event, a configuration control board met to discuss the feedback and developers began fixing or discussing problems with the test event participants. This rapid test approach allowed us to get feedback and work problems right away. Fixes were incorporated into the next test event. Performance test cases were conducted with up to 50 simultaneous users doing the same functions on the force registration application. During this process, we found memory leaks to be causing significant delays and were able to be proactive in optimizing performance before the application went *into the field*. Testing resulted in force registration, the application for unit registration data, performing five times faster when users access the most popular functions of the application. Further application testing led to response time improvements of up to three times. Performance enhancements were made where applications may have many simultaneous users in the United States and abroad.

Figure 1: *Example Agile Process Flow for the Scrum Development Process*



Learned Tenets

Our rapid prototyping and deployment challenge was to take an Army readiness reporting system that had existed for 12 years and modernize, deploy, and support its new capabilities within nine months. This brought about challenges for our multiple contract teams who were beholden to many stakeholders, not to mention being rigidly adherent to changing requirements in a complex wartime environment. These Agile and rapid fielding initiatives led us to several revelations of the rapid prototyping and development approach. We found some key characteristics that are outlined in Table 1.

Rapid prototyping and deployment challenges included development of a new set of capabilities that didn't exist in the old system, and training and re-training a large user community in parallel of building software and testing. New functionality included embedded workflow process, identity management, and Web access using the Secure Internet Protocol Router network. We also had to gain security certificates and authority to operate; because of their generally long lead time, we planned for these issues in the beginning stages. The stakeholders treaded in unfamiliar territory by collaborating closely with users and knowledge sharing with other contractors. Risk management was a collaborative effort that emphasized the software development phase. The greatest challenge was to develop, train, and conduct integration testing with multiple contractors in nine months.

Success

The initial fielding of DRRS-A capabili-

ties were successfully developed, taught to users, fielded, and supported within nine months by using incremental and Agile methodologies [6]. Subsequent releases have been just as successful adding on existing capabilities and deploying new ones. The success of DRRS-A and its net-centric capabilities include improving user accuracy, ease of use, and decreasing manpower and manual input. As an example of cost realization, the USARC cited DRRS-A Web-based applications savings that are expected to be more than $1 million annually, and the Army Reserve Medical Command has saved what averages to be $118,933 per month. The DRRS-A capabilities are part of the Defense Information System Agency's Net-Enabled Command Capability program and readiness model for the U.S. Air Force and Marine Corps.

Conclusion

Lessons learned during program development and fielding have brought the importance of collaboration, communication, and risk management to the forefront of the Agile development process:

- A tight collaboration of several contracting teams, stakeholders, and program offices is necessary to prove integration of the right requirements into the software.
- The set-up of checkpoints during the process is crucial to ensure that development was meeting the customers' needs.
- Without the communication and involvement of stakeholders and customers, there would be limited sharing and transfer of knowledge that can hinder synchronization across the battlespace.
- It is easy to concentrate on risk management in the software development stage since it is the meat of the program, but lessons learned has taught us that all of the other life-cycle stages need to be risk-analyzed and evaluated often during rapid development. For example, training needs to be planned up-front and during development or else there won't be ample time to train the user community. This, in turn, could lead to program failure.

As stated in the introduction, our intent was to give a current successful data point to the DoD's evolutionary acquisition strategy using an Agile life-cycle approach. We have also proven—through cost and user acceptance of this system—that DoD life cycles can be developed and maintained using Agile methodologies. With an aggressive schedule and high program visibility, we broke traditional developmental and cultural barriers by implementing an Agile and evolutionary approach to rapid prototyping, development, and fielding. It was critical to implore a knowledge-sharing environment between contractors and a close collaboration between the functional proponents and users, which in turn laid the groundwork for success. Our Agile and flexible approach to systems and software engineering allowed us to capture the true essence of rapid prototyping and capability deployment while still meeting budgetary, schedule, and customer satisfaction goals.◆

References

1. "Army to Modernize its Unit Status Reporting Processes." Army Stand-To! 10 Aug. 2006 <http://lists.army.mil/pipermail/stand-to/2006-August/000136.html>.
2. DoD. Department of Defense Directive 5000.1: The Defense Acquisition System. Washington, D.C.: Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, 2003.
3. Boehm, Barry, and Richard Turner. Balancing Agility and Discipline: A Guide for the Perplexed. Boston: Addison-Wesley, 2004.
4. "What Is Scrum?" Scrum: It's About Common Sense <http://controlchaos.com/about>.
5. Hansen, W.J., et al. Spiral Development and Evolutionary Acquisition. The SEI-CSE Workshop Special Report. SEI, Carnegie Mellon University. May 2001 <www.sei.cmu.edu/pub/documents/01.reports/pdf/01sr005.pdf>.
6. "New System Gauges Military Readiness." AFCEA Signal. 16 Oct. 2006 <www.imakenews.com/signal/e_000144589000039843.cfm?x=b8fBhsr,b5kS6S>.

About the Authors

Robert Cloutier, Ph.D., is an associate professor of systems engineering in the School of Systems and Enterprises at the Stevens Institute of Technology. He has more than 20 years experience in systems engineering and architecting, software engineering, and project management in both commercial and defense industries. Industry roles include lead avionics engineer, chief enterprise architect, lead software engineer, and system architect on a number of efforts and proposals. His research interests include model-based systems engineering and systems architecting using Unified and Systems Modeling Languages, reference architectures, systems engineering patterns, and architecture management. Cloutier has a bachelor's degree from the U.S. Naval Academy, an MBA from Eastern College, and a doctorate in systems engineering from the Stevens Institute of Technology.

School of Systems and Enterprises
Stevens Institute of Technology
Hoboken, NJ 07030
Phone: (201) 216-5378
E-mail: robert.cloutier@stevens.edu

Portia Crowe is currently working for the Army, Program Executive Office C3T-PM Battle Command as chief engineer of the Army Defense Readiness and Projection Systems. She is conducting research in integrating risk management in Agile systems engineering. Crowe has received two Commander's Awards for Civilian Service: one for successful implementation of the DRRS-A program, and the other for excellence in research and development of technology objectives. She has a bachelor's degree in computer science from Rutgers University, a master's degree in engineering management from the New Jersey Institute of Technology, and is currently a doctoral student in the School of Systems Engineering at the Stevens Institute of Technology.

U.S. Army, PEO C3T-PM
Battle Command
SFAE C3T BC
BLDG 2525-Bay 3
Ft. Monmouth, NJ 07703
Phone: (732) 427-5757
E-mail: portia.crowe@us.army.mil

CROSSTALK
The Journal of Defense Software Engineering

Get Your Free Subscription

Fill out and send us this form.

517 SMXS/MXDEA
6022 FIR AVE
BLDG 1238
HILL AFB, UT 84056-5820
FAX: (801) 777-8069 DSN: 777-8069
PHONE: (801) 775-5555 DSN: 775-5555

Or request online at www.stsc.hill.af.mil

NAME:_____

RANK/GRADE:_____

POSITION/TITLE:_____

ORGANIZATION:_____

ADDRESS:_____

_____

BASE/CITY:_____

STATE:_____ZIP:_____

PHONE:(____)_____

FAX:(____)_____

E-MAIL:_____

CHECK BOX(ES) TO REQUEST BACK ISSUES:

NOV2007 ❑ WORKING AS A TEAM
DEC2007 ❑ SOFTWARE SUSTAINMENT
FEB2008 ❑ SMALL PROJECTS, BIG ISSUES
MAR2008 ❑ THE BEGINNING
APR2008 ❑ PROJECT TRACKING
MAY2008 ❑ LEAN PRINCIPLES
SEPT2008 ❑ APPLICATION SECURITY
OCT2008 ❑ FAULT-TOLERANT SYSTEMS
NOV2008 ❑ INTEROPERABILITY
DEC2008 ❑ DATA AND DATA MGMT.
JAN2009 ❑ ENG. FOR PRODUCTION
FEB2009 ❑ SW AND SYS INTEGRATION
MAR/APR09 ❑ REIN. GOOD PRACTICES

TO REQUEST BACK ISSUES ON TOPICS NOT LISTED ABOVE, PLEASE CONTACT <STSC.CUSTOMERSERVICE@HILL.AF.MIL> .

COMING IN THE JULY/AUGUST ISSUE

Process Replication

From the battlefield to games like Guitar Hero®, processes such as the PSP and TSP are proving successful.

CROSSTALK explores articles in this issue that emphasize the idea that these techniques are an effective and powerful software development methodology when based on best practices. Articles will address process replication efforts, PSP/TSP applications, success stories, lessons learned, and enhancements to repeatable software and non-software practices.

Including the article, "Lean Enablers for Systems Engineering" by Bohdan W. Oppenheim

Look for it in your mailbox early July!

Issue sponsored by:

NAVAIR

A Distributed Multi-Company Software Project

Dr. William R. Nichols, Anita D. Carleton, Watts S. Humphrey, James W. Over
Software Engineering Institute

This article describes how two software development groups—in different locations and working for competing companies—were able to jointly develop a large and complex military systems product. Even with incompatible support systems and decentralized management, these groups were able to work cooperatively and deliver a quality product on schedule[1].

The DoD and almost all non-defense government agencies increasingly depend on large software-intensive systems. Unfortunately, with few exceptions, such large-system development projects are seriously troubled. While the programs may ultimately deliver workable systems, they are generally late, cost much more than planned, and some even fail completely. While guidelines such as CMMI provide useful guidance on what methods should be used, many studies have shown that seriously troubled large-scale development programs have one problem in common: They do not effectively follow the methods they currently know [1].

The two companies doing the work described in this article are the only suppliers the DoD has for a category of highly classified military equipment. These companies' development laboratories were in different cities and had very different engineering practices and support systems. This complicated the DoD's logistics and training systems, raised maintenance costs, and limited acquisition flexibility. The DoD had urged the companies to develop a common engineering design and support system, but the companies had resisted. Finally, the DoD issued a joint contract and directed the companies to develop a common system.

To meet this directive on schedule and within prescribed costs, the senior executives of both companies knew they needed a joint development team that had a level of coordination and communication that they had not previously been able to achieve. Their joint team would have approximately 30 people from two competing organizations with several different disciplines and with different processes and practices. To build such a capability, they turned to the SEI for guidance. The project described in this article is the result.

Project Goals and Strategy

The SEI recommended that the two organizations use the TSP to guide them in jointly establishing the project's goals and objectives, agreeing on the project management strategy, and launching and managing the development work. To start TSP introduction, the SEI had both management teams come to Pittsburgh for a one-day executive seminar and a half-day planning session [2]. The seminar described how the TSP team-building process melds people from different specialties and organizations into high-performing development teams with common goals, practices, processes, and plans. TSP introduction also requires that team leaders take a leadership course and that team members take Personal Software Process℠ (PSP℠) training [3].

Table 1: *The Process Steps in PSP Training*

PSP Steps	Program Number	New Process Concepts Introduced
PSP0	1	Students use their current process with two added measurements. They record time spent per process phase (planning, design, code, review, compile, unit test, and post-mortem). They define a defect type standard, and log all defects found in the review, compile, and unit-test phases.
PSP0.1	2, 3	Students define a coding standard and a line of code (LOC) counting standard, use process improvement proposals (PIPs), and start measuring the size of their programs in LOC.
PSP1	4	Students add defined size estimation methods and effort estimation methods to their personal process. Test reports are also introduced.
PSP1.1	5, 6	Task and schedule planning are introduced. Earned Value (EV) tracking is also introduced.
PSP2	7	Quality techniques are introduced. Structured personal code and design reviews, based on individual defect data, are conducted.
PSP2.1	8, 9, 10	Design templates and design verification methods are introduced.

Team Training

All of the software engineers were trained in common classes where the team members met their counterparts from the other company and completed a series of 10 programming assignments. Writing programs with the six PSP steps taught them to plan and track their work and to measure and manage product quality (see Table 1). The requirements team members were also trained in a single Introduction to Personal Process course that taught them how to plan and track their work.

The introduction strategy started by building self-directed teams that would plan, manage, and track their own work. To be on a self-directed team, however, the members had to learn how to manage themselves. PSP training showed them how to gather data, use that data to make accurate plans, precisely report project status, and manage product quality. After completing training, the team members could see from their personal data that these methods improved their performance (see Table 2).

The first column of Table 2 describes the measure, the second shows the class average value for the first three PSP programs, and the third column shows the class average value of the measure for the last three programs at the end of PSP training.

The TSP Project Launch

Once management and the team members were trained and the teams formed, the next step was to build self-directed teams. This was done in a five-day project launch where the members produced their own plans and negotiated their commitments with management and the customer [4].

℠ Personal Software Process and PSP are service marks of Carnegie Mellon University.

With the TSP, each development team and its team leader is called a unit team; the five-day multi-team launch for this project had three unit teams. Everyone was brought to one location and the leadership team was formed from the three unit team leaders and the two company project managers. It monitored launch progress and helped the unit teams coordinate their work and resolve issues. A TSP coach guided each unit team through the launch and a multi-team coach guided the leadership team and coordinated the overall launch process (see Figure 1).

The First Launch Meeting

In launch meeting 1, the three unit teams, the TSP coaches, and the leadership team met with senior management and customer representatives. The multi-team coach first described the launch process and agenda and two company executives described the product requirements and desired eight-month delivery date. The customer representatives next explained the DoD's reasons for needing a single engineering and manufacturing support system. At the end of meeting 1, all of the team members understood the job, had heard management's goals, and knew why the project was important to the customer. Following meeting 1, the unit teams, team leaders, and coaches met in separate groups with no observers or visitors to follow the launch process for meetings 2 through 8.

Launch Meetings 2 Through 8

In launch meeting 2, the unit teams each established team goals and the members each selected from among the eight standard TSP roles: customer interface manager, design manager, implementation manager, test manager, planning manager, process manager, quality manager, and support manager. By accepting a role or alternate role, each team member took responsibility for an aspect of the team's work. The test managers, for example, did not necessarily do the testing, but they were responsible for ensuring that testing issues were properly considered and addressed throughout the project. The teams decided that no additional roles were required and every team member except the team leader took at least one role or an alternate role.

In launch meetings 3 and 4, the teams each defined their unit team strategies, processes, and plans. They listed the products to be produced, estimated their sizes, and judged the time required for each process step. They then estimated the hours they would each spend on the project every week and generated the project schedule.

Measure	Start of Training	End of Training
Percent time spent in compile	10.8%	1.2%
Percent time spent in unit test	26.3%	10.4%
Compile defect density (number of defects found during compile per thousand LOC [KLOC])	54.0 defects/KLOC	11.7 defect/KLOC
Unit test defect density (number of defects found during unit test per KLOC)	32.0 defects/KLOC	9.5 defects KLOC
Yield (percentage of defects found before first compile)	2.6%	66.5%
Productivity (in LOC per hour)	33.9 LOC/hour	34.0 LOC/hr

Table 2: *Performance Improvement During PSP Training*

Once the unit teams had produced their task plans, they produced a quality plan in launch meeting 5. This included quality-tracking measures, review rates, defect injection and removal rates, yields, and defect levels. The result was an estimate for the defects to be injected and removed in each project phase, the defects in the product at system-test entry, the number of defects that would remain for customer acceptance testing, and the expected number of defects in the product at final delivery.

In meeting 6, the teams made detailed next-phase plans and balanced the workload between the teams and team members. This resulted in the most efficient workload allocation and the minimal project schedule. During meeting 7, they identified and ranked the major project risks and assigned team members to track and prepare mitigation plans for each key risk.

In meeting 8, the teams prepared for the management review in meeting 9. They had been unable to meet the requested eight-month schedule and their base plan delivered a minimum-function first release in 13 months with two subsequent releases at six-month intervals. They also prepared an alternate plan with added resources but, since no suitably skilled and PSP-trained professionals were available, it was considered impractical.

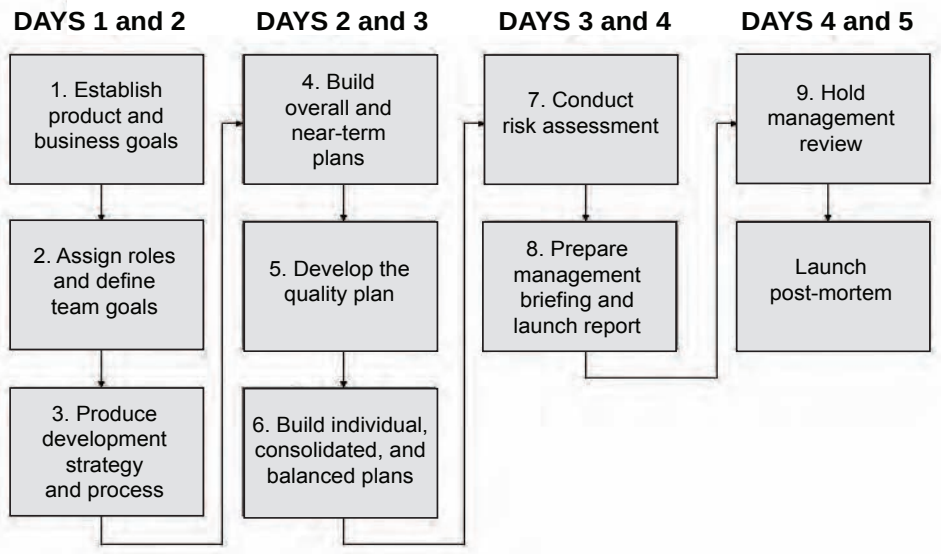
In the post-mortem step, the teams reviewed the launch process, submitted PIPs, and gathered and stored the launch data and materials.

By the morning of the fifth day, the unit team's plans were finished and consolidated into an overall project plan. Each team member had a personal plan, and the overall project had a plan that all team members had participated in producing and would defend as best they could with the available resources. The members were all committed to making the project a success and were excited about the challenges ahead.

Launch Meeting 9 With Management and the Customer

The DoD program manager was invited to

Figure 1: *The Standard TSP Unit-Team Launch Process and Meetings*



the final management meeting with all of the team members, senior management, team leaders, coaches, and involved support-group managers. The unit team leaders first described the planning process and the unit team plans. There was considerable discussion about the program schedule, the release contents, the delivery dates, and how the team had estimated the work.

Everyone was nervous about the program manager's reaction. While he had not said much during the discussion, they knew that he would be hard to convince. Both companies had a long record of missing software commitments and the developers felt that, unless he bought into this plan, they would have to continue with the same practices that had failed in the past. At the end of the plan presentation, the program manager said: "I am disappointed with the dates, but this is the best program plan I have ever seen." He congratulated them on producing a detailed and convincing plan with dates he felt he could count on. While he would have liked an earlier date, he felt it was much more important to have a solid date that he could use to make commitments to the DoD. They had won—now all they had to do was deliver! With a common plan they all believed in, they were excited and ready to tackle the challenges ahead.

Challenges During the Launch

The principal challenge during the launch was to ensure that the unit teams' plans fit together into an overall plan that minimized the program schedule. That required each unit team to identify its dependencies on the other teams and adjust its plan to minimize the overall project schedule. The leadership team met with the team leaders and the four TSP coaches every evening to guide this coordination work. The following three examples show how the TSP launch process helped the teams identify and resolve many of the problems that they would not normally have recognized until later in the program (when those problems would have caused serious delays).

1. The Requirements Schedule Problem. Even though the requirements team members were from different organizations and had a range of skills, they agreed on the team goals and roles in about three hours. Then they drafted a tentative requirements schedule and invited the two development team leaders to a review meeting. Each of the development team leaders had also prepared initial plans that did not mesh with the preliminary requirements plan. They needed some planned requirements items immediately so they could start high-level design work while other items could be deferred until detailed design or implementation. After about an hour, the development team leaders and the requirements team had produced an agreed-upon requirements priority. The requirements team then produced an initial requirements schedule that put the high-priority development items first and gave this plan to the development teams to use in producing their plans.

2. The Role-Manager Teams. Since many activities required a project-wide perspective, cross-team role-manager teams were formed of the role managers from the three unit teams. For example, the planning role-manager team tracked and coordinated the dependencies among the teams. Similarly, the test and quality role-manager teams ensured that testing and quality-management tasks were included in every development phase. The leadership team members mentored the role-manager teams and kept them working on key cross-team issues. For example, the design role-manager team's principal assignment was to ensure that all high-level design work remained consistent and to define common design methods and standards. The manager also ensured that each element of the product work breakdown structure was assigned to the unit team that was best able to handle it.

3. Integrating the Unit Team Plans. After the unit teams produced their plans on launch day 2, the leadership team found that one development team was planning on three releases and the other team on four. After considerable discussion, the leadership team agreed on a strategy with three releases. They also asked the design and customer-interface role-manager teams to produce and maintain a single official list of the functions planned for each release. The development teams then produced their detailed plans and schedules and these schedules were adjusted to synchronize the product release dates. The teams found they could only deliver the minimum required product function in 13 months rather than the desired eight months. The two follow-on releases would then complete the basic product function in 25 months.

Using the Process on the Project

Before the TSP, the managers had made the team plans. Because these were high-level plans, they did not provide the detail needed for precise project tracking, and management had never been able to get accurate project status information. The managers had been reluctant to delegate planning, tracking, and plan management responsibility to the teams; however, after the executive seminar, they agreed that the new training should enable the members to make the detailed plans needed to precisely plan, track, and manage their own work. The unit teams' detailed planning and tracking data enabled them to manage their resources, precisely report status, and get management help with problems before they delayed the project.

Every week, the teams reported planned and actual EV, the Cost Performance Index (CPI), and task hours to the leadership team. With these reports, the leadership team could precisely track and report project status to senior management; in turn, senior management could see if any milestones were exposed and identify and help resolve any problems. For example, achieving a milestone with less EV than planned generally meant that some quality steps had been skipped, while slipping milestones often indicated coordination problems or approval delays.

Part-Time Staffing

Part-time staff assignments were a particularly difficult challenge. While the unit teams had a number of part-time members, several were not even putting in the time they had committed to their teams. Because these part-time members often possessed special knowledge or skill, their work could not be shifted to others. The detailed TSP plans and effort tracking allowed the managers to see this problem in the first few weeks of the job. As the teams gathered data, it was soon evident that, when members planned to work less than half of their time, they often failed to work even that amount. Some critical and highly specialized skills were obtained on a part-time basis, but the team leaders found that many of the other part-time members were costing the project more money than their work was worth. As one team leader said, "I could trade three half-timers for one full-timer and come out ahead."

Using a Defined Process

Many people were initially skeptical about the value added by a defined process. But they soon saw that their previous informal process of circulating documents, marking them up, and having changes incorporated by the document owner did not scale up to a project with numerous stakeholders. Changes could conflict or be contentious, and baseline documents were difficult to identify. A member of the leadership team then created a simple

cross-team process for circulating baseline documents, collecting comments, and distributing the collected comments and revisions for a second review. This worked so well that an additional review was rarely needed. This experience demonstrated that a defined process would help to make the work more effective and efficient and would not add unnecessary bureaucracy.

Another key process element was standards development. Different stakeholders had different document needs. The requirements engineers used technical specifications to resolve analysis issues. Once a technical specification was approved by the customer, it was used by the programmers to design and implement the code. By establishing standards for content, format, and conventions, communication among stakeholders was improved, all needs were addressed, and the review scope could be narrowed. The programmers appreciated a consistent presentation and precise calculations while the engineers received fewer nuisance comments. Eventually, standards were developed for coding, configuration management, design representations, testing, and reports.

Process Improvement Proposals

When the teams realized that they could change the process, they started collecting process problems and analyzing causes before each re-launch. They evaluated their PIPs during re-launch preparation and made process changes before starting the next development cycle. The availability of the PIP mechanism enabled the team members to identify places where they found the process inconvenient or inefficient and then get those issues resolved. This improved process effectiveness and gave the team members a sense of process ownership.

The coaches also reviewed the teams' process fidelity at the seven-week point. They found that all teams were regularly collecting some of their data but that these data were not as complete or accurate as required. This finding suggested that further process changes, as well as some additional team member training and coaching, was needed. The planning managers also started taking a more active role. These actions improved data quality and enabled the team members to better manage their personal plans and to promptly modify their task lists to reflect work changes.

Estimating Improvement

After a year and several cycles of feedback and learning, team estimating accuracy had improved significantly. All teams were now gathering data and meeting and using team data to balance workloads. A team member who had been unable to plan and track at the previous review was now training new project members on planning and making commitments. Plans were more realistic and performance-to-plan was much improved. In the project's initial three-month checkpoint, the task hour plan-to-actual ratio improved from 1.09 to 0.96, and the CPI improved from 0.88 to 0.95 (where a value of 1 means on-time and on-cost performance, respectively). This means that instead of being 35 percent late, they were 3 percent early, and the CPI improved by 7 percent.

Use of Data

Later in the project, the teams started reviewing their data before each re-launch and analyzing test defects. Although the size and defect data were incomplete, the team could see the costs of finding and fixing defects. The most significant finding was that 10 percent of the defects were in the testing materials and another 10 percent were due to incorrect or misunderstood requirements. Coding defects seldom required more than an hour to find and fix while the non-coding defects averaged more than four hours each, and the test-case defects were often even more time-consuming. These and other findings led to improved requirements and design inspections as well as team and leadership workshops on topics shown by the data to need improvement.

Team Problems

Surprisingly, having a distributed team was not a problem for the requirements team. The members built trust by creating common documents, having weekly videoconference status meetings, arranging for regular face-to-face contact, and defining common team goals. The members felt that getting together every other month helped to keep the team jelled.

Even though the development teams were in one location, they initially had not become jelled or smoothly working units. The reason was that management had created an organizational structure with a collocated development team at each company location. This meant that each development team had responsibility for what had originally been thought to be a separate sub-product. After high-level design, however, these sub-products were found to have unforeseen dependencies that required the teams to share a large amount of common code. The teams had not planned for this level of interaction and found it difficult to coordinate their work. The team members had frequent disagreements and team cohesion was soon destroyed.

To solve these problems, the project members proposed an alternate team structure that retained the requirements team but restructured the two software development teams to have members from both companies and locations. Based on the experience of the requirements team, these teams were each given one tightly coupled component to develop and their members were geographically distributed. These restructured teams were established at the next re-launch where they each defined their processes and developed their plans. This structure produced two cohesive, productive, but distributed teams.

Another issue was that as the products matured through subsequent follow-on releases, they required more maintenance work as well as much more regression testing. The requirements team's role shifted from product specification to verification and validation, and the developers spent more time with the requirements engineers on test requirements and expected test results. This led to another project reorganization where the project was reformed into three fully integrated product teams with requirements engineers teamed with the software developers. Because the technical coupling of the work had changed, the new organization was needed so the requirements engineers and software developers could define common goals, share work priorities, have a comprehensive strategy, and agree to interim milestones.

Project Results

The planning, measurement, and quality-management skills the team members gained in PSP training enabled them to handle the large-systems challenges of this project (as shown in Table 2). Without such skills, the individual and team plans would have been incomplete and inaccurate and product quality would have been marginal at best. With accurate plans and consistently high product quality, the three project unit teams consistently met their commitments to each other and to management. As the teams learned to work collaboratively, they learned to anticipate and resolve problems, increasing their productivity and accelerating the work. They completed the originally committed product releases essentially on schedule, though some first-release functions were deferred. The customer was pleased and extended the project to include maintenance and follow-on development. The

organizations have continued to use the TSP process.

Conclusion

Prior to introducing the TSP, the two competing companies described in this article had been unable to produce the joint product for which they had contracted with the government customer. In fact, they had not even been able to agree on a plan for doing the development work. Following the introduction of the TSP, the companies formed a joint team and quickly produced a project plan that the government representative accepted. They then delivered a quality product on schedule.

By using the TSP's self-directed management style and forming integrated development teams, these competing organizations were able to build a collaborative working relationship and predictably produce quality results, even when working at distributed locations and having separate management reporting chains. This experience demonstrates the power of a properly defined, planned, measured, and quality-controlled process in helping organizations produce quality products on predictable schedules. It also shows that a properly formed, suitably trained team with common goals, processes, and plans will overcome the distrust and friction normal with teams from different organizations or separate locations. Without a defined, measured, planned, tracked, and quality-controlled process like the one used by these teams, large-scale distributed development teams have rarely been successful.◆

References

1. Hansen, Marc, and Robert F. Nesbit. Report of the Defense Science Board Task Force on Defense Software. Washington, D.C.: Office of the Under Secretary of Defense for Acquisition and Technology, 2000.
2. Humphrey, Watts S. Winning With Software: An Executive Strategy. Boston: Addison-Wesley, 2002.
3. Humphrey, Watts S. PSP: A Self-Improvement Process for Software Engineers. Boston: Addison-Wesley, 2005.
4. Humphrey, Watts S. TSP: Coaching Development Teams. Boston: Addison-Wesley, 2006.

Note

1. Since the work was highly classified, this article cannot name the application or development organizations.

About the Authors

William R. Nichols, Ph.D., joined the SEI in 2006 as a senior member of the technical staff and serves as a PSP instructor and coach with the TSP program. Prior to joining the SEI, Nichols led a software development team at the Bettis Laboratory near Pittsburgh where he developed and maintained nuclear engineering and scientific software for 14 years. His publication topics include the interaction patterns on software development teams, design and performance of a physics data acquisition system, analyses and results from a particle physics experiment, and algorithm development for use in neutron diffusion programs. He has a doctorate in physics from Carnegie Mellon University.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213-2612
Phone: (412) 268-2727
Fax: (412) 268-5758
E-mail: wm@sei.cmu.edu

Anita D. Carleton is a senior member of the technical staff at the SEI. She has worked for more than 20 years on software process improvement, process measurement, and the TSP. Carleton helped to launch the software measurement initiative at the SEI in 1988. She is the co-author of "Measuring the Software Process: Statistical Process Control for Software Process Improvement." She was awarded with a commendation from Dr. Barry Boehm for her leadership in producing measurement definition frameworks for the DoD Software Action Plan. Carleton has a bachelor's degree in applied mathematics from Carnegie Mellon University and is a senior member of the IEEE Computer Society.

Phone: (412) 268-7718
Fax: (412) 268-5758
E-mail: adc@sei.cmu.edu

Watts S. Humphrey joined the SEI after his retirement from IBM. He established the SEI's Process Program and led development of the CMM for Software, the PSP, and the TSP. At IBM, he managed their commercial software development and was vice president of technical development. He is a fellow for the SEI, the Association of Computing Machinery, and the IEEE. He is also a past member of the Malcolm Baldrige National Quality Award Board of Examiners. In 2003, the President awarded Humphrey the prestigious National Medal of Technology for his contributions to the software engineering community. He holds master's degrees in physics and business administration.

Phone: (412) 268-6379
Fax: (412) 268-5758
E-mail: watts@sei.cmu.edu

James W. Over, who has been with the SEI since 1987, is manager of the TSP and is a senior member of the technical staff for the Software Engineering Process Management Program. Both in the United States and abroad, Over has led the SEI's work transitioning the TSP into several hundred organizations. He received an award from the Boeing Corporation for innovation and leadership in software process improvement. He has 35 years of technical and management experience in the software engineering industry. Over is the co-author of several SEI publications on software process definition and improvement.

Phone: (412) 268-7624
Fax: (412) 268-5758
E-mail: jwo@sei.cmu.edu

Measuring Maintenance Activities Within Development Projects

Lori Holmes and Roger Heller
Q/P Management Group, Inc.

Although function points (FPs) are a good measure of the functionality that is added, changed, or removed through a development project, they do not measure other functions that may be impacted by a specific change but are not actually changed themselves. As well, there is often project work separate from the FP measurable functionality that cannot be counted under the current International Function Point Users Group (IFPUG) 4.2 rules [1]. *This article explores impact points (IPs), a measure which accounts for these issues.*

Function points are an accepted industry standard method used for measuring the size of software projects and applications. They can be used in conjunction with other data for estimating as well as in conducting productivity and quality analyses. Often organizations use FPs and FP-based measures as the key metric in outsourcing contracts.

One drawback, though, is the inability of FPs to measure other functions that may be impacted by a specific change but are not actually changed themselves. Often, additional testing is required downstream of functions that will be using data from changed functions, but they themselves are not modified by the project and are not considered to be within the project scope. This increases effort and can affect productivity.

There is also work separate from the FP measurable functionality that cannot be counted under IFPUG 4.2 rules, including changes to static Web pages or populating code tables that are not related or used by the FP countable functions. The Q/P Management Group recognized, along with our outsourcing clients, that something was needed to measure the impacted functions not covered by IFPUG 4.2 FPs. It was determined that in order to ensure more accurate estimates and to provide a good foundation for productivity measures for use in outsourcing contracts, a separate measure was required. It was important for the measure to be:

- Consistent.
- Repeatable.
- Unbiased to promote good development techniques.
- Easy to apply with little effort and/or expense.

IP Concept Development

Brainstorming sessions were held to explore ideas to support the need for additional measures. The initial brainstorming activity focused on identifying situations when the FP analysis of projects resulted in zero FPs. These situations were then categorized and analyzed to ensure that they met the criteria of zero FP projects and the functionality was unrelated to functions that could be measured with IFPUG 4.2 FPs.

This type of work is often completed by a separate maintenance group and maintenance measures are used (e.g., application FPs/full-time equivalent support staff for a year). However, not all organizations have application FP counts to utilize this measure. In these situations, this work is completed as an enhancement project and needs to be measured separately. Depending on the structure of the organization, zero FP projects can equal between 15 and 20 percent of the organization's development work.

"It was determined that in order to ensure more accurate estimates and to provide a good foundation for productivity measures ... a separate measure was required."

Separate sessions were held to identify potential measures for these non-FP countable situations. One concept focused on a technical approach of counting the number of files or number of tables involved. When this concept was tested, issues arose related to consistency, objectivity, and the number of measures. It seemed as though the resulting size measure would be impacted by how the software was developed. As a result, this approach cannot be used to measure productivity consistently across technologies or for different development methodologies.

The objective of developing the alternative measure was not to measure the technical aspects of a project; it was to measure user-recognizable functions that are impacted by a project but are not changed. In recent years, Q/P has developed a measure for analyzing functionality that needed to be tested in a project, called test points, to estimate required test cases and testing effort. Building on this work and utilizing the standard IFPUG FP methodology, Q/P developed IPs.

While developing this concept, it became clear that IPs have several benefits. IPs are:

- A single measure for all impacted functionality so it is a manageable addition to a measurement program.
- Independent of technology and implementation techniques so it can be used in all development environments.
- Based on the IFPUG FP methodology. Therefore, it does not require an extensive set of guidelines to be developed and does not require extensive training for employees already familiar with FP counting.
- A consistent measure for zero FP projects so they can be used to quantify productivity rates separately from FP-based productivity.

What Are IPs?

IPs account for functions that are impacted but not changed by a project. They follow the same concept as FPs, but focus on non-FP countable projects and functions within projects. It is imperative that IPs only be used for sizing functionality not accounted for under traditional FP analysis. The intent is not to diminish the use of FP measures with overlapping measures but rather to fill a void that exists in FP-based software measurement. Since the IP measure is intended to be complementary to FPs, it is important to account for each separately. Data related to IPs should be

kept in a separate repository from FPs; IP productivity rates should be developed and reported independently from FP productivity rates.

Once a non-countable function is identified, the IFPUG concepts are used to define the function and measure the complexity.

Projects that can be counted using FP analysis are not candidates for IP counts. IP countable items include:

- **Table Updates.** Examples are rate changes, adding products and/or services, and parameter/configuration changes.
- **Code/Text Changes.** Examples are static page updates, Web content updates, cosmetic changes, format changes, sort changes, adding or changing help, or error messaging.
- **Data Management.** Examples are data migration and database restructuring.
- **Technical.** Examples are multiple browsers and new sources of data (e.g., networks).

In all of these, the functions using the new text, updates, etc., would be identified as impacted functions and counted to derive the project's IPs.

Possible Scenarios for Considering IPs

The following are three examples of possible applications of IPs, including one scenario when IPs shouldn't be used.

Example 1: Use IPs

A project requires changes to a logical file to add additional fields necessary for calculating billing rates. Input screens and display screens of these new fields also require changes. In addition, the bill creation process logic must be changed to incorporate the new calculations. All of this functionality would be FP countable.

Once the bill is generated, it is stored in a logical file and the elementary process is completed. All changes for the project are included in the process that ultimately stores the bill. No software changes are required beyond the saving of the bill. However, functions exist that display the past bills and send the bills to other systems. These functions use the file where the bill is stored, but themselves do not require any software changes. Thus, even though they are not modified by the project, these functions are impacted because they must pass along the bill as they did before. These functions do not generate any FPs but they have been impacted by the change and need to be tested to ensure that everything works properly.

The functions would then be counted using the IFPUG definitions to determine if they are external outputs or external inquiries (EQs), and the IFPUG complexity ratings would be applied.

Example 2: Use IPs

A request has been made to add fields to multiple Web pages that retrieve information from developer-maintained files and do not calculate, derive, or maintain any data.

Under IFPUG rules, these functions are not countable because they would be EQs with zero file types referenced (FTR), which is not allowed. Data maintained by developers are not considered as internal logical files or external interface files, so the files containing the Web page information cannot be FTRs.

"It is imperative that IPs do not overlap with the functionality that is measured by FPs. In other words, IPs should not be used to double-count or overstate the amount of work that needs to be delivered."

IPs allow for inquiry functions (EQs) to be counted when there are zero FTRs accessed. The functional complexity would be determined based on zero FTRs and the number of data element types that are entered or displayed.

Example 3: Do Not Use IPs

A project requires new rates to be added to a table. This does not require any change to the table structure, just rows to be added. It is also necessary that logic be changed in a screen to use the new rates and to utilize different calculations when inputting data.

In this case, the table is not countable with FPs, but the screen is countable due to the logic changes to the calculations. For the specific requirement to be delivered, both changes are necessary. The table is just the development technique to assure that the rates are available to the screen. This table change is related to the FP countable change, and any effort related to both activities should be considered together when estimating the project or calculating the productivity rate.

In this case, IPs should not be used for the table change because the change is related to a function that is FP countable.

Testing and Implementation of IPs

To implement IPs in an organization, it is important to assess and define the types of projects and situations that will use them. In addition, to use IPs for estimating and measurement, a baseline study should be conducted to quantify the current situation and establish productivity rates. Some steps to consider are to:

- Develop a list of non-FP countable situations and projects then categorize them by type and volume (e.g., rate changes, new products, etc.).
- Conduct IP counts on a representative sample of projects from the non-FP countable project list.
- Capture effort and delivery platform for non-FP countable projects included in the study to baseline their productivity rates.
- Assess the productivity rates (IPs per hour) to determine trends and any further breakdowns/measures needed.
- Develop templates to use going forward to avoid conducting IP counts on all non-FP countable projects.
  - If rate changes typically impact the same functions, then the same IP count would be used each time.
  - Based on the baseline data, one to three templates may be developed per category (e.g., low, medium, and/or high).

Conclusion

IPs can be a useful tool for organizations that have a large amount of non-FP countable projects or portions of projects. They can be used similarly to FPs in measuring productivity and quality for these projects.

IPs are not intended to replace FPs, but are meant to provide a supplement for the areas FPs do not cover. It is imperative that IPs do not overlap with the functionality that is measured by FPs. In other words, IPs should not be used to double-count or overstate the amount of work that needs to be delivered.

IPs, as a measure for non-FP countable items, are beneficial for the following reasons:

- It provides one measure for all non-FP countable projects.
- Once an impacted function is identified, guidelines for how to count are already available (IFPUG).
- The measure is not impacted by the development techniques or how things are physically implemented.
- Organizations can associate productivity rates for each appropriate segment (e.g., platform, type of change, size, etc.).
- Organizations can develop IP templates for each type of non-FP countable project to reuse on future projects of the same type.

Q/P is confident that IPs will fill the void that organizations have with measuring non-FP countable projects. Using IPs correctly will provide consistent measurement data that can be used in estimating, productivity, and quality analyses, as well as in outsourcing contract negotiations.◆

Reference

1. IFPUG Counting Practices Committee. Function Point Counting Practices Manual. Release 4.2.1, Jan. 2005.

About the Authors

Lori Holmes is a director with Q/P, specializing in software measurement, process improvement, and quality assurance. Her areas of expertise include FP analysis, software project estimation, establishing measurement programs, and software quality assurance. Holmes is recognized as an international consultant, speaker, and instructor. She focuses on helping organizations implement quality and productivity improvement programs utilizing measurement techniques. She is an experienced instructor in change management, quality inspections, measurement, and FP analysis.

Q/P Management Group, Inc.
10 Bow ST
Stoneham, MA 02180-1343
Phone: (410) 544-5781
Fax: (410) 544-5827
E-mail: lori.holmes@qpmg.com

Roger Heller is Q/P's vice president. He is a recognized consultant, instructor, and speaker who specializes in helping organizations improve software quality and productivity through measurement. Heller has assisted numerous organizations in establishing software measurement programs and quality improvement initiatives. His other areas of expertise include FP analysis (and its application techniques in emerging environments), evaluating and measuring technical architectures, software estimating, and application and technology planning.

Q/P Management Group, Inc.
10 Bow ST
Stoneham, MA 02180-1343
Phone: (941) 629-0943
Fax: (941) 629-2467
E-mail: rheller@qpmg.com

Web Sites

Tilcon IDS White Paper

www.tilcon.com/manual/Tilcon_WhitePaper.pdf

The Tilcon Interface Development Suite (IDS) is a multi-platform user interface development solution (graphical user interface/human-machine interface builder) that delivers robust, highly interactive user interfaces for real-time, embedded, and mission-critical applications. Readers will get a high-level technical overview of the Tilcon IDS, details regarding the compiler, processor, and operating systems supported, and a quick overview of how IDS works "from the user's perspective." You can also learn more about the main components—the interface builder, the embedded vector engine, and the application programming interface—as discussed in this issue's article, "Using WYSIWYG GUI Tools With UML."

Assessment of the Ballistic Missile Defense System (BMDS)

www.cdi.org/pdfs/FY08BMDSJan09.pdf

As discussed in this issue's article "Software Safety for Model-Driven Development," the Missile Defense Agency's Command, Control, Battle Management, and Communications program has been integrating elements of the Global Engagement Manager (GEM) software development methodology into its BMDS. This January 2009 report—from the Director of Operational Test and Evaluation—looks back at the BDMS in 2008. It includes an examination of recent progress with respect to baselines and goals, a determination of the current test program's adequacy, and an analysis of the system's overall operational effectiveness. This report also gives a GEM progress assessment and discusses their future expectations for the GEM.

The Defense Readiness Reporting System: A New Tool for Force Management

www.ndu.edu/inss/Press/jfq_pages/editions/i39/i39_forum_05.pdf

In this edition of CrossTalk, Portia Crowe and Dr. Robert Cloutier show "Evolutionary Capabilities Developed and Fielded in Nine Months," with the Defense Readiness Reporting System-Army (DRRS-A) at the heart of that success. Now Laura J. Junor—director and scientific adviser for the Office of the Secretary of Defense Readiness Programming and Assessment Division—explores the DRRS-A, which moves the focus of force managers from reporting unit readiness to managing force capabilities. In this *Joint Force Quarterly* article, Junor explores the DRRS-A in regards to how force managers can adapt to meet current needs under the DRRS-A, residual capability identification, what capabilities the DoD has and how they can be tailored and combined to respond to operational needs, and the roll of the Web-based Enhanced Status of Resources and Training System.

Function Point Basics – IFPUG 4.2

www.totalmetrics.com/__data/assets/pdf_file/0012/2046/IFPUG-4.2-Quick-Reference.pdf

In this issue's "Measuring Maintenance Activities Within Development Projects," Lori Holmes and Roger Heller discuss a way to measure impacted functions not covered by the International Function Point User's Group (IFPUG) Counting Practices Manual (CPM) 4.2. If you are foggy on the ins and outs of IFPUG or CPM 4.2, or just want a refresher, Total Metrics has simplified the concepts down to an easily understood two-page "cheat sheet." Several IFPUG 4.2-specific terms are defined: user, user view, user identifiable, application boundary, scope, and control information. You can also learn the basics of external input, external output, external inquiry, internal logical files, external interface files, function points awarded (FPA), and the FPA measurement process.

Watts New?

www.sei.cmu.edu/news-at-sei/columns/watts_new/watts-new.htm

Watts Humphrey, co-author of this issue's article, "A Distributed Multi-Company Software Project," has his own column with the SEI called *Watts New*? As perhaps the best-known member of the SEI's technical staff, Humphrey has taken readers on a process-improvement journey, step by step. Also available, at <www.sei.cmu.edu/news-at-sei/columns/watts_new/watts-new-compiled.pdf>, is a collection of his columns dating back to 1998. Humphrey explores a range of topics: the problem of setting impossible dates for project completion, planning as a team, using the Team Software Process, the importance of removing software defects, applying discipline to software development, and approaching managers about a process improvement effort.

A View of 20th and 21st Century Software Engineering

http://sunset.usc.edu/csse/TECHRPTS/2006/usccsse2006-626/usccsse2006-626.pdf

In this issue's Open Forum, "From Substandard to Successful Software," Martin Allen recognizes Barry Boehm's call to look at history for successes—instead of just failures not to repeat. In this sweeping report covering Boehm's 54-year career in software engineering, he identifies some of the past major software experiences that are worth repeating (as well as some that are not). Boehm explores software engineering's past, present, and future: the 1950s, including the semi-automated ground environment software development process; software crafting in the 1960s; formality and the Waterfall processes of the 1970s; the productivity and scalability of the 1980s; concurrent versus sequential processes in the 1990s; our current focus on agility and value; and the future, analyzing how globalization and systems of systems might affect the next decade.

Software Technology Support Center (STSC) Consulting Services

www.stsc.hill.af.mil/consulting

Does your organization need help implementing and appraising CMMI? Do you struggle with software estimation? Do you need help assessing, preparing, planning, applying, and effectively using software technologies? Do you want to learn more about requirements engineering technology, object-oriented programming, Ada for managers, or the Personal Software Process? STSC Consulting Services—a U.S. Air Force-based provider of assessment and workshops—is the first line of defense in solving software challenges in your organization.

From Substandard to Successful Software

Martin Allen
Independent Software Consultant

Our global finance industries are in the grip of a fearsome tempest known to us as the credit crunch, triggered by bad subprime mortgages and toxic debt. Is there a lesson here for our industries and government agencies that have become reliant on software-intensive systems of systems (SISoS) and are susceptible to the potential ravages of inferior software? It is essential for the software industry to identify and tackle what I call substandard software: software life-cycle products that do not have basic quality attributes such as reliability, usability, accuracy, efficiency, adaptability, and testability. This article provides indicators and professional advice in a set of seven rules that will increase the probability of a successful software project.

In 2007, a financial earthquake started shaking the stability of global economies, with the epicenter situated in the subprime mortgage market. Banks and investment institutions with hundreds of years of displaying prudence disclosed that their very foundations were riddled with toxic debt. From this, the credit crunch was born. The scale of the problem was enough to bankrupt countries such as Iceland[1], and behemoths in the banking industry continue collapsing or are subject to nationalization. Given the tens of thousands of financial experts employed and the maturity of risk management processes in the credit industry, one question remains: How did the problem become so large and so widespread without earlier detection?

Very few of us are qualified to understand why the circumstances for economic meltdown were not avoided, nor are the majority of our citizens equipped to comprehend even the basics of the global banking industry and the risks in the credit systems. Nonetheless, some of us ask whether there is a warning in the credit crunch for the acquirers and providers of high-technology systems. Could unmanageable levels of substandard software paralyze the software industry the way that unmanageable levels of subprime loans paralyzed finance?

Substandard Software

Recently, we have witnessed a significant rise in the number of organizations relying heavily on the successful deployment of SISoS. In industries such as defense, transport, medical, communications, energy, space, entertainment, and finance, reliance on software keeps growing. An exponential increase in the magnitude and complexity of the systems attempted is also evident. Undoubtedly, the risks from substandard software have increased respectively. For instance, the volume and complexity of software systems in the average family car have increased substantially in the last 10 years; however, it is clear that the vast majority of these software systems are dependable because otherwise our roads would be strewn with inoperable vehicles. On the other hand, there have been a few well-publicized hiccups with embedded software in the automotive industry, culminating in the embarrassing and expensive recall of thousands of vehicles [1]. A glut of substandard software is certainly capable of damaging the reputation of a car manufacturer, and is theoretically capable of contributing to the demise of automotive giants. Articles such as "Software's Chronic Crisis" [2] and various other reports suggest that multiple industries are afflicted by substandard software.

> *"Far too many substantial projects still flounder when involving software-intensive systems."*

Far too many substantial projects still flounder when involving software-intensive systems. Project success in terms of cost, schedule, capability, and user acceptance is too rare an occurrence (refer to the sidebar on page 31). Furthermore, significant numbers of software systems that are successfully deployed are later found to have issues with maintainability, portability, scalability, flexibility, and reliability. In part, this is due to the fact that the first thing to be sacrificed in the rush to deploy a late project is quality; this provides a perfect incubator for substandard software. Professional engineers are taught to sacrifice non-essential system capabilities first and quality last. If the constraints of a schedule or budget mean an *all-singing-and-all-dancing* system cannot be deployed, then at least it may still satisfy customers and users temporarily.

Causal Factors

One favored approach for participants in failed or troubled software projects is to perform a forensic analysis or lessons-learned exercise. From experience, this tends to inappropriately concentrate an organization's attention on symptomatic rather than causal factors. By way of analogy, take the medical profession: For many decades, they strived to move from treating symptoms to curing disease to disease prevention; treatment of symptoms is often necessary, but far from sufficient.

It is easy to understand why a systems acquirer or supplier team struggles to overcome inherent bias in order to focus on and expose fundamental technical and managerial weaknesses. An alternative for them is to work diligently on eradicating symptomatic factors on succeeding programs, while ignoring conveniently the endemic causes of project problems. In the wake of an expensive project failure, there often follows an inglorious frenzy to reinvent software engineering processes from first principles, or the latest project management tool is rolled out in true panacea fashion. Thereby, unsuccessful project teams are provided with a myriad of opportunities to spread the spores of substandard software into other areas. Most diseases and infections rely on some interaction of their human hosts in order to spread and multiply; likewise, the spread of substandard software depends on people. In short, lessons-learned initiatives aimed at organizational improvement, based solely on failed or troubled projects, may be simply acting as a Trojan Horse[2].

The Seven Rules for Success

Analyzing and reporting the causal factors of failing software projects is often too

onerous or uncomfortable an assignment for organizations. A complementary and more palatable strategy may be to concentrate on the fundamental factors that professionals know contribute to successful teams and projects. This is the transpose of a typical lessons-learned initiative. Could such an approach be influential in halting a substandard software plague[3]?

The following is a proposed set of seven rules to enhance the probability of success:

- **Rule 1:** Professionalism and software engineering competence should be assessed objectively and encouraged proactively by senior management.
- **Rule 2:** The number and seniority of software professionals employed within an organization should be commensurate with the magnitude and criticality of the required software systems.
- **Rule 3:** Good quality life-cycle products are the essential ingredients in the development, deployment, and maintenance of successful software systems.
- **Rule 4:** Mature industry standards are key to the production of high-quality, consistent life-cycle data products.
- **Rule 5:** Employ COTS software products with due care.
- **Rule 6:** The formality or weight of processes needs to be tailored and applied in accordance with project risks.
- **Rule 7:** Processes are necessary but not sufficient; competent people and practical life-cycle products are equally necessary.

In the following sections, the origins of these rules are considered in the context of the three elementary components of a project. I refer to these as the 3Ps: people, products, and processes.

People

In "The Mythical Man-Month," Frederick P. Brooks suggested that a tenfold productivity chasm existed between the most and least efficient software development teams [5]. This hypothesis was given further credence in Barry Boehm's "Software Engineering Economics" [6], and by subsequent studies. From assembly languages to Java, from structured methods to computer-aided software engineering tools, from process improvement to object orientation, each advance in software engineering has been fundamentally reliant on its application by competent managers and engineers (Rule 1). My extensive experience also helps confirm that a professional and competent staff, organized efficiently, is the primary influence on productivity.

Enterprises that depend on software systems should assess carefully and periodically whether the levels of education, training, experience, and seniority of personnel can accommodate the current and future needs of their industry. If anything is a causal factor in the success of projects, it is individual and team competence.

For some project teams, learning a best engineering practice is perhaps easier than abandoning systemic poor practices. When current project teams are found to be designing with flow charts, professionals must question why four decades of engineering progress are ignored. In "The Challenges of Complex IT Projects," professionalism and education are identified as having a major influence:

> A striking proportion of project difficulties stem from people in both customer and supplier organizations failing to implement known best practices. This can be ascribed to the general absence of collective professionalism in the IT industry, as well as inadequacies in the education and training of customer and supplier staff at all levels. [7]

Increasingly, there are calls for the competence of personnel working in safety-critical industries to be assessed and managed [8, 9]. At the time of writing this article, there were no legislative requirements in place in Europe to regulate the competence of software safety professionals.

What differentiates professionals and amateurs? Their behavior. Software professionals are characterized by a relevant education and continued learning; they have a holistic or system life-cycle focus; they work to industry standards; and they have a balanced approach to technical risk. In contrast, amateurs have no relevant software engineering education: They focus on implementation, coding, and tools, and have a naïve view of risk.

In some situations, organizations employing software professionals still find success elusive. This may be due to the pattern of seniority within the teams. In an environment where deployment of SISoS is essential to the business, software professionals should hold a commensurate level of senior roles (Rule 2). One possible alternative to the proliferation of process maturity models (e.g., CMMI) could be a framework for objective assessment of an organization's capability, based on the number and seniority of competent software engineers employed.

Along with the seniority of competent personnel, organizational structure is known to have a significant bearing on sustained success. The relative advantages and disadvantages of project-managed and discipline-managed structures have been acknowledged for many years. A hybrid matrix management structure is a reputable alternative that attempts to capture the advantages of both while reducing the disadvantages. Experience has shown that the matrix works effectively when the (software engineering) discipline is responsible for strategic decisions, while tactical decisions are best made by project teams. For example, the selection of a system architecture is a strategic concern. Organizational policy may be used to separate explicitly strategic and tactical concerns.

Products

If processes are viewed as the recipes for development of successful software systems, then life-cycle products are the vital *ingredients* (Rule 3). Good quality ingredients are essential in the creation of gastronomic delights, and there is a direct analogy here with software products. There is a finite limit to what can be achieved by processing or cooking with inferior ingredients.

Several mature software standards, particularly those originating from the DoD and IEEE, emphasize the production of data items (Rule 4). Software professionals recognized many years ago that the collection and management of cohesive, decoupled life-cycle data or information products is a crucial facet of the discipline. Tangible life-cycle data are required to support both verification and validation activities or processes.

Complementary ingredients are required to produce a decent meal, just as the availability of good quality code does not by itself satisfy the multiple criteria for a successful software system. Typical software system life-cycle products include:

- ConOps.
- System requirement specification.
- System architectural design.
- Software requirements specification.
- Interface requirement/design.
- Software architectural design.
- Source code.
- Executable code.
- A qualification test plan.
- A qualification test description.
- A development plan.
- A quality and configuration plan.

Superior standards tend to be prescriptive in the types of life-cycle data required, generic in the way the life-cycle data is collected and managed, and flexible enough to support tailoring for different categories of software projects. Effective tai-

loring is arranged from safety-critical, through mission- or business-critical, down to support software. Additionally, several standards provide evaluation criteria to direct the quality control of life-cycle products.

One proposed approach to the global software crisis—prevalent now in the defense communities—is to employ COTS software products (Rule 5). This remains a commendable strategy but is far from a panacea. There remain significant issues with the deployment of COTS-based applications into environments for which they were not intended (as indicated in [10]).

Processes

Software system processes and organizational process maturity have received a significant degree of academic and industry exposure in the 1990s and 2000s. The ubiquitous CMM and CMMI offerings from the SEI are familiar to software engineers around the globe. As stated previously in this article, processes are the essential recipes for the development of software systems.

However, it is a popular myth (cautioned in [10]) that processes alone can transform an enterprise from a Level 1 ad-hoc underachiever to a Level 5-optimized corporate machine. An established tenet is that overemphasis on processes and procedures makes an organization bureaucratic, inflexible, and overly reliant on checklists.

It has been suggested that the extraordinary interest in Agile methods has been fueled in part by a general dissatisfaction with heavyweight processes; the proponents of processes have fractured into two opposing entrenched camps: CMMI versus Agile. Quite simply, the weight or formality of processes applied should be commensurate with the project risks, especially with respect to magnitude and criticality (Rule 6). This is an established approach with safety-related standards such as DO-178B and IEC 61508. Such issues are also tackled in [10]. A measure of diligence is required because processes and techniques that are considered successful on small developments do not tend to scale-up. Software standards MIL-STD-498 and IEEE 12207, which are geared toward medium to large-scale SISoS development and integration, use project risk criteria as keys to the suitability of different life-cycle models. Grand design, incremental, and evolutionary models are included in these standards.

To all the proponents of software process—whether light or heavy, Agile or CMMI—a note of caution is appropriate. Stringent processes are very effective in volume manufacturing environments (from where the original concepts of process and quality control originated) where repeatability and consistency are gods. Stringent processes are also effective in software engineering. However, software engineering can never resemble a production line, and processes are necessary but not sufficient for success because competent people and practical life-cycle products are similarly indispensable (Rule 7).

3Ps Aligned

Ensuring the alignment of the 3Ps by employing these seven rules is no guarantee, but it will maximize the probability of success. I have had the privilege of working on and witnessing several successful projects where competent people—analysts, designers, project managers, test professionals, and programmers—employed mature processes to produce dependable software products.

Commentators have remarked that many financial institutions will survive the present credit crunch, but will necessarily emerge with a different *modus operandi*, particularly with respect to acceptable risk. Perhaps too, our industries that are dependent on SISoS may have to go to the edge of a similar abyss before taking the opportunity to address the causal factors of substandard software. Alternatively, influential people and organizations could start applying the seven rules and may never need to explain why so many experts overlooked the substandard software epidemic.◆

Substandard Software Indicators

In 2007, I was involved in a large European collaboration project with three international customers and one main system supplier worth hundreds of millions of Euros. It was an environment that lent itself to creating substandard software. The main indicators were weaknesses in:

- **Standards Compliance.** The supplier organization had no demonstrable evidence of compliance on previous or current projects.
- **The Life-Cycle Model.** The supplier initially proposed the use of Agile methods, even though the application was large and safety-related [3].
- **Concept of Operations (ConOps).** After more than two years of project activity, the ConOps had not been considered.
- **Software and System Qualification Testing.** Contrary to all mature standards and test regimes, qualification testing did not feature in any project plans.
- **Data Modeling.** The system was predominantly a data management and distribution system, but no data model was available.
- **Competence.** The customers and supplier lacked competent software professionals in the project management teams.
- **System and Software Architecture.** Techniques to derive the system architecture were archaic and discredited (e.g., functional decomposition).

The lack of ability was a causal factor, whereas the other factors were symptomatic.

References

1. Noon, Chris. "Okuda's Toyota Recalls Prius Fleet Over Software Glitch." Forbes. 14 Oct. 2005 <www.forbes.com/2005/10/14/toyota-prius-recall-cx_cn_1014autofacescan03. html>.
2. Wayt Gibbs, W. "Software's Chronic Crisis." Scientific American. Sept. 1994.
3. Boehm, Barry W., and Richard Turner. Balancing Agility and Discipline: A Guide for the Perplexed. New York: Addison-Wesley Longman, 2003.
4. Boehm, Barry W. A View of 20th and 21st Century Software Engineering. Proc. of the 28th International Conference on Software Engineering. Shanghai, China: 20-28 May, 2006.
5. Brooks, Frederick P. The Mythical Man-Month: Essays on Software Engineering. New York: Addison-Wesley Longman, 1974.
6. Boehm, Barry W. Software Engineering Economics. Upper Saddle River, NJ: Prentice Hall PTR, 1981.
7. The Royal Academy of Engineering and The British Computer Society. The Challenges of Complex IT Projects. London, UK: The Royal Academy of Engineering, Apr. 2004.
8. Health and Safety Executive, The Institution of Electrical Engineers, and The British Computer Society. Managing Competence for Safety-Related Systems. 2007 <www.hse.gov.uk/humanfactors/comah/mancomppt1.pdf>.
9. International Electrotechnical Commission: IEC 61508-1. Functional Safety of Electrical/Electronic/Pro-

Coming Events

June 29-July 2
ACM SIGMOD/PODS 2009
Providence, RI
www.sigmod09.org

July 1-3
21st International Conference on Software and Knowledge Engineering
Boston, MA
www.ksi.edu/seke/seke09.html

July 13-16
2009 International Conference on Software Engineering Theory and Practice
Orlando, FL
www.promoteresearch.org/2009/setp

July 19-23
International Symposium on Software Testing and Analysis
Chicago, IL
www.cse.msu.edu/issta09

July 20-24
33rd Annual IEEE International Computer Software and Applications Conference
Seattle, WA
http://conferences.computer.org/compsac/2009/

August 18-20
LandWarNet Conference
Ft. Lauderdale, FL
www.afcea.org/events/landwarnet/09

August 24-28
13th International Software Product Line Conference
San Francisco, CA
www.sei.cmu.edu/activities/splc2009/

2010
22nd 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.**

grammable Electronic Safety-Related Systems. Appendix B. 1 Dec. 1998.
10. Boehm, Barry, Peter Kind, and Richard Turner. "Risky Business: 7 Myths About Software Engineering That Impact Defense Acquisitions." Project Management. May-June 2002 <www.dau.mil/pubs/pm/pmpdf02/boe-mj2.pdf>.

Notes

1. There are many articles on this topic, including <www.cnn.com/2009/WORLD/europe/01/26/iceland.government>.
2. This is the more classic definition of "Trojan Horse" (see <http://en.wikipedia.com/wiki/Trojan_Horse>), not the computer malware.
3. Barry W. Boehm, the respected software engineering professor at the University of Southern California, hints at such a strategy in "A View of 20th and 21st Century Software Engineering." In a riposte to George Santayana's famous quote, "Those who cannot remember the past are condemned to repeat it," Boehm advises that failing to acknowledge and record past successes condemns an organization not to repeat them [4].

About the Author

Martin Allen is a software engineering professional with more than 28 years experience, mostly in the defense industry in the United Kingdom. He has worked on many successful software intensive systems for the British Royal Air Force and the Royal Navy. Allen has always had a strong interest in industry standards for the engineering of dependable systems. His other professional interests include risk management, software cost economics, requirements analysis, design methods, and software testing. Allen and his colleagues work on the boundary between the academic research of computer science and the practical application of software engineering.

37 Priestfields
Fareham, Hampshire
United Kingdom, PO14 4TE
Phone: +0044 (0) 7881542031
E-mail: mjallen60@yahoo.co.uk

Letter to the Editor

Dear CrossTalk,

As someone who supports process engineering in my job, and in light of the numerous and recent cuts to process improvement (PI) efforts across the nation, I felt the need to share my thoughts with the PI community.

Most PI professionals soon realize that their job is unique in that they not only need to perform all of the activities related to PI itself, but that they also need to be able to sell the value of what they do.

In many organizations, PI is often viewed with just a bit of suspicion, especially by those in management who see it as "overhead," or just a fad that will eventually fade away. When times get tough and the belt-tightening starts, PI can become an easy target for an organization needing to reduce or redirect resources. The harsh reality is that in this job we need to always remember that there is a "target" on our back. We not only need to produce value, but show that value on a regular basis—or risk becoming another statistic.

Some time ago, my organization decided that we (and by inference me) would no longer perform compliance auditing of the more than 200 projects in our IT department. My first impression was that this was simply a "less than ideal" decision by management, but upon reflection, I realized that the root of the problem was me: My team and I had been plugging along doing some really terrific work, but failing to regularly showcase our successes before key stakeholders. We showed a lot of data, but not to all the right people and with little emphasis on our unique value. In short, we had become complacent with *doing* the work and forgot about the essential element of *selling* it.

So what's a PI professional to do? How do you improve the chances that your work isn't the next target for a reduction? If I could go back in time and give myself some advice, here is what I would have proposed:

1. Stakeholder management is the lifeblood of PI efforts. Identify your key stakeholders and work to uncover their real needs. Refresh and re-evaluate this information on a regular basis. Over time, stakeholder roles and needs will change, and (by extension) so will their perceptions of the value of your work.
2. Find a "champion": someone who understands what you do and has influence in the organization. In many cases, you may need to educate them on why your work is so important and let them see the actual results. If you can involve your champion in what you are doing in such a way that they can put their fingerprints on your work—and even claim some of the credit for your results—all the better.
3. Showcase your successes regularly and seek out feedback from those you support. Feedback from tools such as mini-customer satisfaction surveys can help, but actual quotes from your customers are even better. If management knows that you are making their people more productive, they are more likely to put your work above the "cut" line.
4. Think "small words and large fonts" when presenting your value to stakeholders. If it's simple and focused on the bottom line, they are more likely to comprehend and remember it (and you).
5. Bring data: how many dollars you saved the organization, the impact of the risks you helped them to avoid, etc. An interesting activity is to take a look at your own metrics set from the standpoint of the value and viability of your work: Do these metrics tell a compelling story of your value to your key stakeholders?

In these tough economic times, our employers need more of what PI professionals do, not less. Our ability to help them work "better, faster, cheaper" is even more critical now. We need to constantly be aware that it's not only our job to deliver the improvements, but to keep management aware of the value and the need to continue on this path.

—Terry Leip
IT Program Management Office, Intel Corporation
Terry.Leip@intel.com

Call for Articles

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

21st Century Defense
November/December 2009
Submission Deadline: June 12, 2009

Getting a Handle on CMMI
January/February 2010
Submission Deadline: August 16, 2009

Please follow the Author Guidelines for CrossTalk, available on the Internet at <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on software-related topics at any time, along with Letters to the Editor and BackTalk. We also provide a link to each monthly theme, giving greater detail on the types of articles we're looking for at <www.stsc.hill.af.mil/crosstalk/theme.html>.

Alan Smithee: Where Are You When I Need You? (or, "A Software Engineer Goes to the Movies")

As a software engineer, I often find myself "on the road," consulting at a customer site. Inevitably, I find myself stuck in a hotel in a strange town, often for several days at a time. Being a fun-loving, thrill-seeking, typical software engineer, I make an effort to find the REALLY fun spots: an electronics store, a bookstore—and, in many cases, a video store or movie theater.

When seeing a movie, I have rather unique criteria: I want the movie to be either really good, or REALLY bad. I don't mean bad as in within 15 minutes you can't remember the title or plot. I mean BAD as in "I'll remember this movie forever. I'll joke for years about it with anybody who had the misfortune to see it also. I'll brag about how horrible it was to all of my friends."

As I was recently watching yet another contender for the dubious honor of REALLY BAD movie, I realized that several of these terrible films I have watched over the decades shared a common trait—they were directed by "Alan Smithee."

I thought it odd that anybody who could make so many "memorable" movies didn't seem to start doing better as a director over the years. As soon as the movie was over, I started searching various Internet movie databases for enlightenment. I found out that Alan Smithee is well-known for his ability to make stinkers. In fact, that's ALL that he makes.

It seems that Alan Smithee (or Allen Smithee – the name varies) is not a real person. To quote from the Internet Movie Database (IMDB)[1]:

> The Directors Guild contract generally does not permit a director to remove her/his name from films ... striving for decades to establish the director as the "author" of a film, and part of getting the credit for the successes is taking the blame for the failures. The only exceptions they make are cases in which a film was clearly taken away from a director and recruit heavily against her/his wishes in ways that completely altered the film. Directors are required to appeal to the Guild in such cases. If the appeal is successful, their name is replaced by Alan Smithee. So if you notice a film directed by Alan Smithee, it is certain it is not what its director intended, and likely that it is not any good.

Wow ... what a great deal! A director can work for several years on a movie, and if he or she decides that somebody else has messed up their "artistic vision," they can have their name removed from the credits.

Imagine, fellow software engineers, that if you are associated with a colossal failure, and the failure is not because of what you have done but because of what others have done to you, you can have your name removed. Imagine if the following problems could be exempted from your responsibility:

- Totally lacking, too much, or poor quality upper-level management.
- Funding cuts.
- Requirements changes that you weren't consulted on.
- Key personnel removed in the middle of the project.
- No requirements at all.

Do the best you can, and submit an appeal to a board of fellow software engineers. If the appeal is upheld, your name would be removed from the project, and a pseudonym used instead[2]!

By the same token, some movie stars, for various reasons, have decided not to take credit for success! Kathleen Turner provided the voice of Jessica Rabbit in "Who Framed Roger Rabbit?" back in 1988. She elected to be unlisted in the credits[3]. However, neither I nor any other software engineers I know have had the problem of being associated with a colossal success, yet wishing to remain anonymous.

I don't think my suggestion of adapting pseudonyms for managing software will be received seriously. Maybe we don't need them. Back in the early '70s—when I was a programmer at Strategic Air Command Headquarters—I remember that my project manager had the following posted on his office wall:

THE FIVE PHASES OF PROJECT MANAGEMENT

1. Initial enthusiasm
2. Inevitable problems
3. Search for someone to blame
4. Punishment of those who are innocent
5. Praise and reward for the non-participants

The project manager in question had been on the job for more than 20 years, and the copy he had posted was very weathered and worn. Obviously, the semi-humorous idea was not new, even then.

According to both the IMDB and the authors of the semi-parody book, "Directed by Allen Smithee[4]," the use of the pseudonym started in 1969.

Come to think of it, maybe Hollywood got the idea from us.

—David A. Cook, Ph.D.
Principal Member of the Technical Staff
The AEgis Technologies Group, Inc.
dcook@aegistg.com

Notes

1. See <www.imdb.com/name/nm0000647/>. While formally discontinued by the DGA in 2000, Alan Smithee seems to live on.
2. I am not suggesting any particular name AT THIS TIME. One of the reviewers jokingly suggested calling such a project a "Dave Cook." I immediately suggested his name instead. Feel free to e-mail me with a similar suggestion, and I'll add YOU to the list of potential names.
3. She only provided the speaking voice. Amy Irving provided the singing voice.
4. See <www.upress.umn.edu/Books/B/braddock_directed.html>.

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.