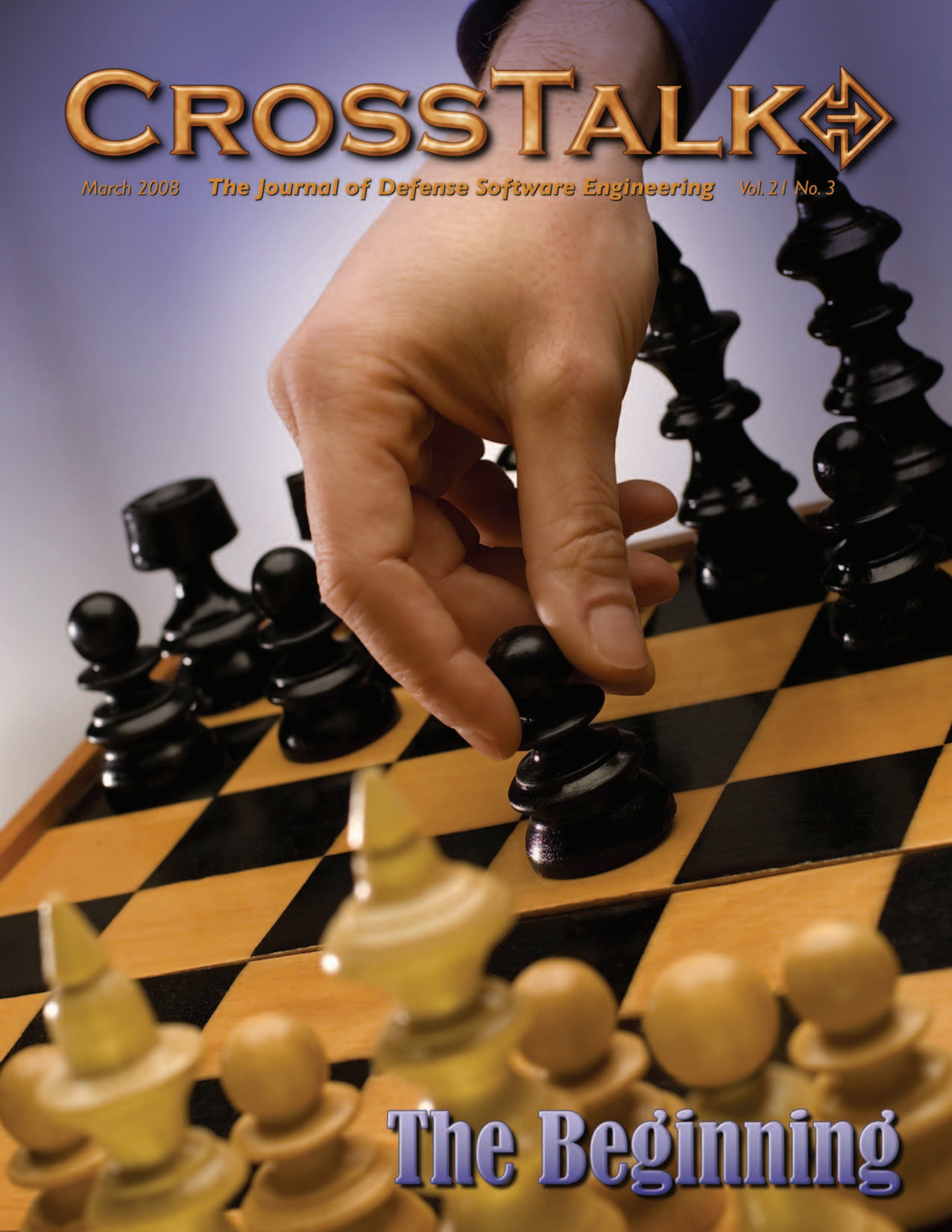
CrossTalk

March 2008 **The Journal of Defense Software Engineering** Vol. 21 No. 3



The Beginning

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **MAR 2008**	2. REPORT TYPE	3. DATES COVERED **00-00-2008 to 00-00-2008**

4. TITLE AND SUBTITLE **CrossTalk: The Journal of Defense Software Engineering. Volume 21, Number 3, March 2008**	5a. CONTRACT NUMBER 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) **OO-ALC/MASE,6022 Fir Ave,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 **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**32**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

On The Cover
Cover Design by Kent Bingham

Additional art services provided by Janna Jensen

CrossTalk

Co-Sponsors:	
DoD-CIO	*The Honorable John Grimes*
OSD (AT&L)	*Kristen Baldwin*
NAVAIR	*Jeff Schwalb*
76 SMXG	*Kevin Stamey*
309 SMXG	*Norman LeClair*
DHS	*Joe Jarzombek*
Staff:	
Managing Director	*Brent Baxter*
Publisher	*Elizabeth Starrett*
Managing Editor	*Kase Johnstun*
Associate Editor	*Chelene Fortier-Lozancich*
Article Coordinator	*Nicole Kentta*
Phone	(801) 775-5555
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/crosstalk

CrossTalk, The Journal of Defense Software Engineering is co-sponsored by the Department of Defense Chief Information Office (DoD-CIO); 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). DoD-CIO co-sponsor: Assistant Secretary of Defense (Networks and Information Integration). OSD (AT&L) co-sponsor: Software Engineering and System Assurance. USN co-sponsor: Naval Air Systems Command. USAF co-sponsors: Oklahoma City-Air Logistics Center (ALC) 76 Software Maintenance Group (SMXG); and Ogden-ALC 309 SMXG. DHS co-sponsor: National Cyber Security Division of the Office of Infrastructure Protection.

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

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 17.

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. Articles published in CrossTalk remain the property of the authors and may be submitted to other publications.

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

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

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

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

Planning for Software Acquisition, Development, and Sustainment in a Complex Systems Environment

As the demand for software acquisition and development expertise continues to rise in an environment of limited resources, we must ensure that programs engage software expertise early in the system life cycle and that the program has tools and processes in place that enable distributed teams to work effectively across geographic boundaries. Best practices highlight the importance of taking the necessary planning in *The Beginning* of a project to properly design a way to reach the desired outcome.

Acquisition of software-intensive systems in today's rapidly evolving technological environment is a challenging task. Consistency of processes for estimating scope, size, and complexity of software is often lacking. Requirements are often unstable or incomplete and are not adequately allocated down to the software domain, the results of which may include excessive rework, cost, and schedule overruns, as well as poor quality products.

We know that up-front planning helps teams save money and deliver a better product to the customer. Through various methods teams consistently use requirements to generate the tasks necessary to create the product. This issue of CROSSTALK focuses on ways to maximize the conversion of project requirements to an effective task plan.

When you read Ellen Gottesdiener's article *Good Practices for Developing User Requirements*, take note of how she dissects an effective approach for defining user requirements. For a deeper understanding of various methods of state machine-based design to perform software development read Markus Herrmannsdörfer, Dr. Sascha Konrad, and Brian Berenbach's *Tabular Notations for State Machine-based Specifications*. Interoperability and secure data sharing in a real-time operational environment are discussed in Dr. Douglas C. Schmidt, Dr. Angelo Corsaro and Hans Van't Hag's article *Addressing the Challenges of Tactical Information Management in Net-Centric Systems With DDS*. Learn how the technical process and people came together successfully on a geographically distributed project in *NAVAIR's Coast to Coast Support of the E-2C Hawkeye Using Distributed TSP* by Linda Lou Crosby and Jeff Schwalb. Also included in this issue of CROSSTALK is experimental data gathered by Dr. David J. Coe and his University of Alabama students leading to recommendations for improving the consistency of the estimation process in *Improving Consistency of Use Case Points Estimates*.

As software assumes an ever-increasing role in the acquisition, development, and sustainment of evolving capabilities within Department of Defense-fielded systems, the need for robust up-front planning of software related activities, processes, best practices and events as an integrated component of the overall system throughout the entire systems engineering life cycle is vital to program success. The future holds great promise. With accelerating technology and great planning we will have a skilled, trained, workforce that shares a common vision and operates in an integrated fashion to achieve a clear set of goals. Effective planning at the outset of every project is crucial to meet the challenges of the future.

Joan Johnson

Joan Johnson
Naval Air Systems Command

NAVAIR's Coast-to-Coast Support of the E-2C Hawkeye Using Distributed TSP

Linda Lou Crosby
NAVAIR People, Process, Products, and Resources Team

Jeff Schwalb
NAVAIR Systems/Software Support Center Team

This article discusses the Naval Air Systems Command (NAVAIR) distributed team that became an example of how projects can work together remotely and be successful. This was accomplished through the use of common processes extended to multiple, distributed teams, their coaches, and the tools used to automate those processes. The other vital area addressed was the organizational cultures to be connected. This meant an important understanding of each other's assumptions, values, and styles needed to happen. This article is filled with observations and lessons learned from team members, team coaches, and organizational facilitators on the multi-site virtual merging of NAVAIR E-2C Hawkeye software developers at Patuxent River, MD with F-14D software developers at Point Mugu, CA.

The challenge facing the E-2C program at Patuxent River in Maryland in 2003 was one of simply having more work than the engineers could perform in the time allotted. At the same time, engineers at Point Mugu in California were working on the F-14D delivering its final block release to the fleet. When E-2C leadership discovered the available pool of engineers at Point Mugu, the question became one of how to successfully combine these two groups of engineers into one distributed team.

E-2C leadership at the time was aware of the F-14A/B/D model aircraft sun setting and the fact that many talented software engineers were becoming available for other work. The F-14 Integrated Product Team at Point Mugu saw working with E-2C as an opportunity to place their software engineers into a team with a bright future. F-14 leadership briefed the E-2C leadership on the capabilities of these software engineers as part of their effort to find a future home for them. Their Team Software Process[SM] (TSP[SM]) credentials were so impressive that the E-2C program decided it was worth the extra effort involved in having virtual teams employed on their upcoming software development projects.

Members from the two sites became two integrated teams, one for the E-2C Mission Computer (MC) and one for the displays on board. The E-2C Leadership asked the NAVAIR Process Improvement (PI) enterprise team for an approach to establish this virtual software engineering team. They would start with TSP to establish a process engineering framework due to its success with other NAVAIR projects [1]. You will hear about three things in this article: TSP and its ability to support multiple, distributed teams, cultural change and how people were supported as the distributed team started, as well as responses about how this effort evolved and what they think of it now as they still continue to work together five years later.

TSP

To start, we will provide a quick review of basic TSP followed by the extensions of the multiple, distributed team version applied to E-2C. The basic TSP is a software engineering process framework created by the Software Engineering Institute (SEI) to help a team plan its work and then work that plan through collection of measures, regular communications, and replanning at milestones along the way to delivery of products [2]. The fundamentals are displayed in Figure 1.

The TSP starts by building a common language between software engineers on a team by training them in the Personal Software Process[SM] (PSP[SM]) that they will each use [3]. This training uses both lectures and exercises so that engineers gain knowledge and experience in the use of the process scripts they will use, collection of basic measures used, and derivation of metrics from those measures so that project plans and actuals can be brought together to have quantitative project status on a weekly basis.

The beginning of the real project work is the launch [4]. It is a set of nine very structured and detailed planning meetings that start by communicating with project stake-

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

F-14D Tomcat – United States Navy's primary maritime air superiority fighter, fleet defense interceptor and tactical reconnaissance platform from 1974 to 2006.

E-2C Hawkeye – Navy's all-weather, carrier-based tactical battle management airborne early warning, command and control aircraft for the Carrier Strike Group and Joint Force Commander.

holders to obtain needs, wants, and desires. From this the team will proceed with identification of roles, goals, products, and services. Then they proceed with the top-down and bottom-up plans necessary to have each member of the team own a balanced workload of tasks that allow two to four tasks to be accomplished each week. This way, when the team becomes operational after launch each person is able to know if they are making progress or if they need to shift tasks with their teammates.

While Figure 1 shows a launch happening at requirements time, a project may actually start TSP anywhere in its life cycle based upon the next opportune time to do so. For example, if a project wishes to apply TSP for the first time and is currently developing requirements, then it will obtain the TSP training sometime shortly before the high-level design phase and then launch after requirements are complete.

Another feature of TSP is planning an entire project from top-down at the beginning and then re-planning as milestones along the way are reached. This is because the level of detailed planning done in TSP should not exceed three to six months due to reasonable horizons of work being done and the idea of working from milestone to milestone. While Figure 1 shows a simple waterflow model, a team may instead choose other strategies where example project cycles develop iteratively functional versions of a project. A typical multi-year project will go through several cycles of bottom-up planning as it moves from one milestone to the next.

Distributed Team

To recap, the E-2C program was doing two things with TSP. It was using *multiple* project teams to deliver its product and in virtual teams that were *distributed* between Maryland and California. Each project team is self-coordinated, with each member acting in one of several technical, support, or lead roles that coordinates all these efforts. Each of the E-2C project's leads, planning coordinators, and quality coordinators would come together in key parts of a multi-team launch. Planning coordinators came together before and after top-down and bottom-up plan meetings to check status and test any assumptions. Quality managers meet after the quality plans have been generated to do the same. Also, at the end of each day of launch the leadership team, consisting of each project lead and the coaches, convene to check status and discover any horizontal issues that may affect each other.

Shown in Table 1 is recent data from the E-2C distributed team plans. Teams were constructed based upon expertise and interest of engineers. The E-2C teams as shown have a good handle on their plans and products. These teams effectively planned their work every four to six months to the level of granularity as described previously. These launches were conducted with the smaller portion of the team typically traveling to allow the entire team to get together. This face-to-face planning style was vital to maintaining trust in the virtual team.

Operationally these teams know where they are with ongoing weekly communications via teleconference. These teams chose to break up their weekly communications. The first is a TSP data-driven meeting to track progress, as shown in Table 2 (see page 6). The other weekly teleconference meeting is used to discuss technical issues. The key is that these teams are planning their work and then working those plans for constant improvement.

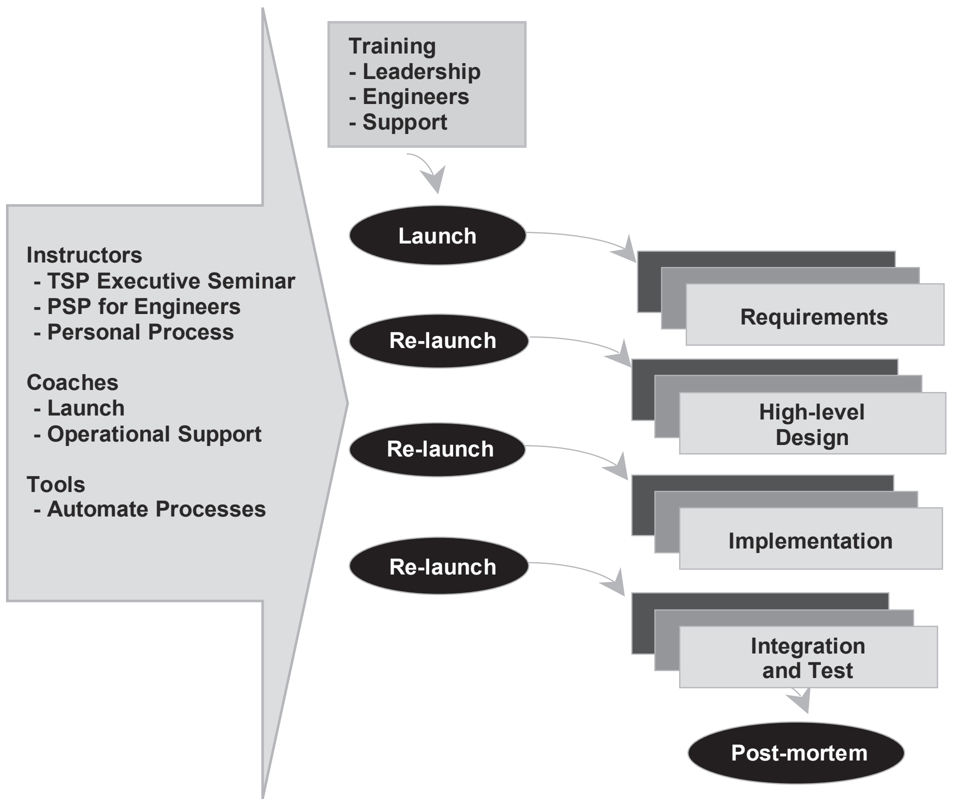


Figure 1: *TSP Approach of Training, Planning, and Operations*

Cultural Change

To address the culture change needed to join the two teams, the NAVAIR Organizational Development (OD) team would work the people issues as the PI team focused on processes. NAVAIR sites had traditionally been perceived as competitors to each other; this is a difficult barrier to break down as many of our working systems still support this perception. Also, the folklore within each site includes stories of past competition. We needed new stories of successful collaboration to replace the competition stories. This team saw the possibilities and we built on that.

Part of the challenge in this effort was the culture ingrained at each NAVAIR site. A Software Support Activity (SSA) maintains and delivers the software needed to bring high tech capabilities to today's advanced aircraft. Without software, the aircraft would be

Table 1: *E-2C Distributed Project Teams*

	Team Members		Project						
Project Type	Pax	Point Mugu	Tasks	Weeks	Tasks/ Eng/ Week	Cycle Complete Date	Hours (planned/ actual)	Earned Value (planned/ actual)
A	8	4	780	16	4.06	12/2006	1.97	1.37
A	8	4	800	24	2.78	7/2007	0.95	1.22
B	4	12	1788	24	4.66	12/2006	1.13	1.18
B	4	12	1713	24	4.46	5/2007	1.12	1.06

A=Advanced Control Indicator Set B=MC

- ❑ What's happening outside the project?
- ❑ Each role coordinator reports
- ❑ Goals
- ❑ Risks
- ❑ Project status (plans vs. actuals)
- ❑ Upcoming tasks and special events

Table 2: *Weekly TSP Team Meeting Typical Agenda*

unable to deliver today's precision weapons. Traditionally, members of an SSA were co-located because all had to be close to the simulation/test lab where they produced/modified the software and tested it. For example, F/A-18 and AV-8B at China Lake, F-14D at Point Mugu, E-2C at Patuxent River, MD, and so forth. With the advances in technology, the availability of high-speed communications lines has gone up and costs have lowered, allowing virtual teaming to become much more common. If anything, it is now the culture – norms, customs, traditions – that seem to stand in the way of increasing the use of virtual teams.

E-2C leadership announced from the start that the teaming arrangement would not be one of developer and subcontractor, but rather a single integrated team – a true partnership. One benefit from this partnership is increased resources. Pax needed more engineers and Point Mugu needed more work. Without that relationship, they would not have been able to give the fleet all the wanted and needed functionality – the benefit being that E-2C would generate more work for itself and help the program grow. In the end, the program thought it could be used as an example to follow when considering a successful multi-site team.

According to the OD team, the challenge was pretty clear: NAVAIR has been producing software-intensive products for decades so the knowledge for software exists with the people all across the NAVAIR sites. Bringing teams together, rather than hiring new people into a site, is more efficient because the cost of recruitment and training is not needed. Instead, the investment is made in building a team rather than in training in the software domain. The people who came together on this team already had the NAVAIR knowledge and knew how to develop good quality software. They just needed to learn how to work together from across the country.

Team Building

Building the virtual teams was accomplished with two initial events. The first was a three day initial gathering in June 2003 designed to start the building of a new common culture. Its objectives were the following:

1. Get every team member to meet and greet, get to know each other, and have some fun.
2. Share history of each subgroup and establish a vision of the future for this newly formed team.
3. Establish team operating principles and obtain team agreement on basic operations.
4. Identify communications methods and processes for initial team operations.

To meet these objectives, the first day was conducted as a set of outdoor activities to get everyone to know each other and have fun. Activities were conducted in a park on base at Pax and included various games such as the following:

- Celebration of success – developing ways to do so.
- Reflection – what in your past will contribute to success.
- Picture cards.
- Ah-So-Koh circle.
- Newspaper talk – sharing information.
- Climbing wall.
- Hula hoop lift.
- Reflection – personal plan, etc.

Table 3: *Individual Feedback*

Project	Location	Comments
Mission Computer	Point Mugu	Team building was valuable: "Although some distrust levels were still around after the team building event, the event lay the foundation for the groups to build a functional team to achieve the common goals."
Foreign Military Sales	Point Mugu	Had his doubts initially but realized E-2C was serious: "…when I saw the effort going in at PAX to provide the training, tools, and resources necessary to get the job done here at Point Mugu, I knew management was really supporting this."
Mission Computer	Point Mugu	Results were the key: "After successfully delivering many projects within schedule for the Version-5 fleet release, I realized that the *distributed approach* was going to work. If people did not work together as a team to solve problems, they simply could not achieve such results. Since it was the first project, working together to deliver Version-5 was the most difficult. Several projects after that were flying smoothly."
Display	Pax	Attributes team success to TSP. Has been a team member since March 2004, develops requirements and detailed design documentation: "TSP provides organization and communications; as a developer, you know exactly what is expected from you from the start of the project. Both managerial and team expectation. In order to accomplish those objectives, you need to have strong communication within the teams."
Mission Computer	Point Mugu	Technology was an issue: "I think the biggest challenge was and is operating a classified network across the country. Not so much because the technology is not there, but because of all the security hoops that we have to go through to get our network approved."
Leadership	Pax	Technology also helped: "Technology aided in allowing this team to work together. We were able to establish a network across country, which allowed the use of a common data repository and common processes to be used. For example, everyone at both sites used the same configuration management system."
Display	Pax	Had previous experience on a distributed team, and did not like it: "It was not well coordinated and I always felt like we were the *poor-stepchildren* in the process." This time the approach was completely different: "There is high coordination and management attention to the issues involved technically in making it work smoothly. I know that this time I am on the big side (East Coast) and so that may make things different, but I think that there is much more sense that the West Coast folks are *real* team members, not just hired help."
Mission Computer	Point Mugu	Importance of communicating across the sites: "The biggest challenge was communication. Several conference calls and meetings between the two sites took place. Several visits were made by team management so they could know every team member and build the bridge between them. These efforts definitely helped."

The second and third day events were conducted indoors with the goal of increased understanding through historical and present-day perspectives. Many of the outcomes of the games and adventures of the first day would be available for use in this second and third day of team building. Activities included the following:

- Team introductions.
- Team history.
- E2C lab tour.
- Myers-Briggs Type Indicator workshop.
- Strengths/weaknesses/opportunities/constraints chart developed by the team.
- Gap analysis determined, solutions proposed, and actions assigned to team members.
- Team members built joint vision for the future.
- Team members drafted team agreement, mission, and vision.

The second event was performed about eight months into the projects in February 2004. A follow-up with E2-C project teams was performed by conducting confidential interviews at both sites. From topics that emerged, a set of team-building topics were presented to team leaders for possible follow up. One of the most impressive discoveries from the confidential interviews was that communication between the two sites and team members was going well – a big plus!

Observations

While engineering process and cultural change were important in making this E-2C multi-distributed team get started, we were most interested in the people themselves and what they thought. With evolving requirements and launches accordingly, these projects still exist and operate in very much the same distributed way as they started nearly five years ago. While that says a lot, we wanted to know what real participants said (see Table 3).

A member of one E-2C team did a good job showing some of the fundamental places where PI and cultural change (see Table 4) took place. Individuals of a team located in different places must know and trust each other to plan their work and then track it. To do so, historical data must be collected and used for tracking and improved planning.

Conclusions

It is important to understand that real people are the key to any technology improvement being successful, especially when it is a distributed team. The important thing for readers to realize is that their situation could accomplish the same great success with the buy-in of people from their organization.

1. **Have a project plan.** Everyone should know the mission and goals. Each member should know who is responsible for what and when.
2. **Learn about available resources from each other.** How many developers are available and what skills or talents does each individual have? What equipment and tools are there for development and testing?
3. **Communicate with each other and communicate often.** Plan weekly meetings, plan face-to-face meetings, e-mail, and call often.
4. **Trust each other.** Team members should respect and understand each other.
5. **Share information.** Team members should share what they know and what they learn with everyone else.
6. **Work to your plan and goals**.

Table 4: *Six Factors That Produce Success for a Multi-Site Team*

The immediate result was the F-14D engineers were given a new lease on life, while E-2C welcomed some incredibly well-versed and knowledgeable people into their program. Long-term results (continued excellence in delivering software products to the Fleet) show that people with similar training and skills can move *laterally* in an organization and continue to make a solid contribution. Finally, the overall experience shows that two separate organizations must remember the importance of considering cultural factors when bringing teams together.

As for the future, it is full speed ahead, and more of the same for the E-2C multi-site team. With initial concerns a thing of the past, the E-2C team can fully focus on the *Hawkeye* mission. E-2C is right on target and they set a great example for others to follow in proving that *miles don't matter* when it comes to having a successful Integrated Product Team.◆

References

1. Wall, Daniel S. "Case Study: Accelerating Process Improvement by Integrating the TSP and CMMI." Pittsburgh, PA: SEI, 2007.
2. Humphrey, Watts S. "TSP: Coaching Development Teams." Addison-Wesley, 2006.
3. Humphrey, Watts S. "PSP: A Self Improvement Process for Software Engineers." Addison-Wesley, 2005.
4. Humphrey, Watts S. "TSP: Coaching Development Teams, Part II Launching a TSP Team." Addison-Wesley, 2006.

About the Authors

Linda Lou Crosby has chronicled software process and organizational improvement at NAVAIR since 1999. Previously, she produced and reported for KCET Public Television in Los Angeles (including an Emmy nomination), wrote an award-winning column, and has created award-winning videos for the U.S. Navy. Crosby is presently working with the Center for Risk Communication and the People, Process, Products, and Resources team at NAVAIR.

NAVAIR
People, Process, Products, and Resources
Code 41E000D
1900 N Knox RD
BLDG 1494 MS 6308
China Lake, CA 93555-6106
Phone: (760) 377-5001
E-mail: Ll_neon@iwvisp.com

Jeff Schwalb is employed by NAVAIR at China Lake, California. Currently, he works in an enterprise team that helps provide continuous process improvement support across NAVAIR. Schwalb has taught each of the TSP/PSP courses many times and has been involved in the TSP launch of several projects across NAVAIR. He is now working with SEI to extend TSP practices into other domains. Schwalb received his bachelor of science degree in computer science from California State University, Chico in 1986.

NAVAIR
Systems/Software Support Center
Code 414300D
1900 N Knox RD
BLDG 1494 MS 6308
China Lake, CA 93555-6106
Phone: (760) 939-6226
E-mail: jeff.schwalb@navy.mil

Improving Consistency of Use Case Points Estimates

Dr. David J. Coe
University of Alabama, Huntsville

Use cases document key product functionality and have been used as the basis for estimating software product development efforts. Presented here is experimental data on the use of the Use Case Points (UCP) method to estimate development effort for a semester-scale software product. From a common set of initial requirements, six student teams refined these requirements and produced independent effort estimates from their own use case models. The sources of estimation inconsistency are examined and recommendations for improving the consistency of the estimation process are presented.

Use cases are a common technique for documenting software requirements. A *use case* is often defined as a step-by-step description of the interaction between the software and one or more *actors* (the people or other systems that utilize the software product to complete a given task). A use case model of a software product is thus the set of all use cases that describe the product's desired functionality. Well-written use cases concisely summarize product functionality in a way that is easy for customers to understand. They may also provide early insight on project complexities and give software developers a starting point from which to estimate total development effort. The UCP estimation method was proposed by Gustav Karner as a way of estimating resources for projects developed using the Objectory methodology (which later evolved into the Unified Process) [1-2].

The UCP method takes into account the complexities of the use cases themselves and the complexities of the users (or actors) that will interact with the software product. A weighted equation based upon the number of steps in each use case and the complexities of the actors determines an initial UCP estimate. This initial estimate is then scaled by a technical factor, to adjust the estimate by the product's perceived technical challenges, and by an environmental factor, to adjust the estimate for people-related factors such as skill levels, experience, motivation, etc. The scaled UCP estimate may then be used to estimate development effort by multiplying it by an experimentally derived Productivity Factor (PF). The mechanics of the UCP calculation resemble that of Albrecht's Function Point method from which this method was derived [2-3].

Students in an introductory software engineering course were required to use Karner's UCP method to estimate the overall effort required to implement a simple hotel reservation system. An overview of the team project is presented, followed by a step-by-step review of the UCP method. The PFs achieved by the student teams are determined using the UCP estimates and time sheet data, and sources of discrepancies in the estimates are analyzed. Finally, the lessons learned from this experience are summarized.

Team Project Requirements

Six, four-person student teams were each given one semester to implement a simplified hotel reservation system in C++ using a common set of initial requirements. The hotel reservation system was to allow hotel employees to make reservations for customers and to generate reports that summarized the current state of the hotel's room reservations. The base rate charged for each hotel room would be set in advance and could be changed to reflect variations in demand throughout the year. The actual rate charged, however, could vary depending upon which of the four types of reservations (prepaid, advance, conventional, or incentive) was selected. The reservation system also had to apply specified penalties for no-show guests. At check out, guests were to receive a bill that summarized the details of their reservation and the total charges.

The reservation system was also required to maintain records of all transactions, including reservation changes and cancellations, and archive those records to disk. Finally, employees were to have the option of generating one of the five types of reports that summarized various aspects of hotel operation, including lists of expected arrivals and current occupancy lists. Additional details on the hotel reservation system project may be found in Appendix A of [3].

Teams were required to practice object-oriented analysis and design techniques learned in the course. Given the time constraints of the semester, a graphical user interface for the reservation system was optional. All teams were required to follow an iterative and incremental development process based upon the five core disciplines of the Unified Process (requirements, analysis, design, implementation, and test). During the project, teams were required to generate and refine seven different deliverables as follows:

- Software Project Management Plan.
- Use Case Model.
- Unified Modeling Language (UML) Class Diagrams.
- UML Sequence Diagrams.
- Test and Integration Plan.
- C++ Source Code and Makefile.
- User Manual.

Five intermediate milestones were established to encourage teams to start the project early and avoid the last minute heroic efforts at the end of the semester. At each milestone, teams were also required to submit timesheets that summarized the total time spent by each team member on the submitted artifacts at that milestone. This data on actual effort would later be used to compute the PF for each team.

I first came across the UCP estimation method in [2] as the teams were nearing completion of their Use Case models. Since my previous software estimation experience utilized line-of-code (LOC) based methods, I added a UCP estimate requirement to the team project to see how well this estimation technique would work, especially since all six teams would be producing estimates for the same product. Students were asked to use their existing use case model, as is, when completing the UCP estimate.

Brief Review of the UCP Method

This overview of the UCP estimation method is derived from [1, 2]. Included in the discussion is an example UCP calculation, utilizing values reported by one of the student teams. Following this review of the UCP method, the estimates prepared by the six student teams are presented and analyzed.

Step 1: Unadjusted Use Case Weight

Starting with the use case model, the first step in the UCP estimation method is to calculate the Unadjusted Use Case Weight (UUCW), which is a weighted sum of the total number of steps identified in all of the use cases. The use cases are sorted into three different categories (Simple, Average, or Complex) depending upon the number of steps (or transactions) identified or the perceived complexity of implementation (estimated number of objects required). Table 1 describes the use case category criteria.

Consider the data reported by Team A. Team A identified a total of 14 use cases for the team project. Seven of these use cases were determined to be *Simple* in that the use cases consist of three or fewer transactions, and seven were considered to be *Average* in that they consisted of four to seven transactions. Team A identified no *Complex* use cases. Thus, the weighted sum UUCW for Team A is computed as follows:

Team A UUCW = 7 * 5 + 7 * 10 + 0 * 15 = 105

Use Case Category	Category Description		Category Weight
	Transactions T per Use Case	Objects Required for Implementation R	
Simple	$T < 4$	$R < 5$	5
Average	$4 \le T \le 7$	$5 \le R \le 10$	10
Complex	$T > 7$	$R > 10$	15

Table 1: *Use Cases Complexity Categories and Weights* [1-2]

Actor Category	Category Description	Category Weight
Simple	Actor represents another system interacting through a defined application programming interface	1
Average	An actor which represents interaction with another system via a protocol or human interaction through a text interface	2
Complex	An actor which interacts through a graphical user interface	3

Table 2: *Actor Complexity Categories and Weights* [1-2]

Step 2: Unadjusted Actor Weight

As with the use cases, the actors are also categorized by their perceived complexity (Simple, Average, or Complex). The Unadjusted Actor Weight (UAW) is a weighted sum of all actors appearing in the use case model. Table 2 summarizes the actor category criteria and category weights. Team A reported a single Complex actor in their use case model and no Simple or Average actors. The weighted sum UAW for Team A is calculated as follows:

Team A UAW = 0 * 1 + 0 * 2 + 1 * 3 = 3

Step 3: Unadjusted UCP

The Unadjusted UCP (UUCP) is computed as the sum of the UUCW and the UAW. For Team A, the UUCP is computed as follows:

Team A UUCP = UUCW + UAW = 105 + 3 = 108

Step 4: Technical Complexity Factor

The Technical Complexity Factor (TCF) is used to adjust the UCP estimate based upon the perceived technical complexities of the project. The influences of 13 technical factors on the development effort are estimated using a scale from 0 (irrelevant) to 5 (essential) with the value 3 used if the factor's influence is unknown. The technical factors and their relative weights are described in Table 3. For each technical factor, the influence estimate is multiplied by the corresponding factor weight and summed across all thirteen factors to compute the Technical Total Factor (TTF). The TCF is computed from the TTF as follows:

TCF = 0.60 + 0.01 * TTF

Using Team A's influence estimates as

Table 3: *Technical factors that influence complexity as described in* [1, 2] *along with their relative weights. Also included are influence estimates of each factor made by Team A. Influence estimates range from* 0 *(irrelevant) to* 5 *(essential).*

Technical Factor	Factor Description	Factor Weight	Team A Influence Estimates
T_1	Distributed systems	2	0
T_2	Response time or throughput performance	1	4
T_3	End user efficiency	1	3
T_4	Complex internal processing	1	2
T_5	Reusability of code in other applications	1	0
T_6	Ease of installation	0.5	4
T_7	Ease of use	0.5	4
T_8	Portability	2	0
T_9	Changeability	1	3
T_{10}	Concurrency	1	0
T_{11}	Special security features	1	0
T_{12}	Provide direct access for third parties	1	0
T_{13}	Special user training facilities	1	2

Table 4: *Environmental factors that influence complexity as described in* [1-2] *along with their relative weights. Also included are influence estimates of each factor made by Team A. Influence estimates range from* 0 *(irrelevant) to* 5 *(essential).*

Environmental Factor	Factor Description	Factor Weight	Team A Influence Estimates
E_1	Familiarity with Unified Process*	1.5	1
E_2	Part time workers	-1	4
E_3	Analyst capability	0.5	1
E_4	Application experience	0.5	1
E_5	Object-oriented experience	1	3
E_6	Motivation	1	4
E_7	Difficult programming language	-1	0
E_8	Stable requirements	2	3

*changed from original term Objectory

Project	Number of Actors (complexity)	Number of Use Cases (complexity)	TCF	ECF	UCPs	Person-Hours Required	PF (Hours per UCP)
A	5 (average)	10 (average)	1.00	0.975	107.25	2150	20.05
B	5 (average)	50 (average)	1.00	1.175	599.25	12500	20.86
C	5 (average)	15 (average)	1.00	1.175	188.00	5400	28.72

Table 5: *Project Metrics Used by Karner to Determine PF* [1]

shown in Table 3, Team A computed a TTF of 18 resulting in a 0.78 TCF.

Step 5: Environmental Complexity Factor

The Environmental Complexity Factor (ECF) is used to adjust the estimate for people-related factors such as skill levels, experience, motivation, etc. The influences of eight environmental factors on the development effort are estimated using a scale from 0 (irrelevant) to 5 (essential) with the value 3 used if the factor's influence is unknown. Table 4 (see previous page) lists the UCP environmental factors, their relative weights, and their estimated impact according to Team A. For each environmental factor, the influence estimate is multiplied by the corresponding factor weight and summed across all eight factors to compute the Environmental Total Factor (ETF). The ECF is computed from the ETF using the following equation:

$$ECF = 1.4 - 0.03 * ETF$$

Using Team A's influence estimates as shown in Table 4, Team A computed an ETF of 11.5 resulting in a 1.06 Environmental Complexity Factor.

Step 6: Compute UCPs and Estimated Labor

The UCP estimate for the product is computed as the product of the initial unadjusted use case points, the TCF, and the ECF. For Team A, the UCP computation is summarized next:

$$\text{Team A}\ \ UCP = UUCP * TCF * ECF = 108 * 0.78 * 1.06 = 89$$

The Estimated Labor is the product of the UCP estimate and an experimentally determined PF. For the three products presented by Karner in [1], the PF was experimentally determined to be in the range of 20-30 person-hours per use case point as shown in Table 5. Fitting a straight line to this data, Karner computed a nominal productivity of 20 person-hours per use case point.

In practice, there are a number of different methods that one might use to determine a PF. One could use UCP PFs documented in the software engineering literature. These values, however, may not produce an accurate estimate since they do not necessarily reflect the types of systems you are developing, the programming languages or development tools that you will be using, etc. Another approach would be to utilize your own organization's project metrics to calculate a PF. An advantage of this approach is that these metrics would account for your particular organization's software development process. Another advantage is that you could selectively include metrics from relevant projects only.

Having no experimental data on student teams using this method for a semester-sized student project, students were asked to complete their estimates using the nominal value of 20 person-hours per use case point to demonstrate that they understood the mechanics of the calculation. Students were warned that this would likely result in an overestimate of the actual labor required on the assigned project since Karner's nominal productivity was derived from data collected from larger commercial products. I planned to use this first classroom experience with UCP estimation to gather metrics that I might use to compute a more realistic PF for subsequent classroom use. My goal was that each student would need to average at minimum five hours per person per week to satisfactorily complete the team software project. Assuming four students per team working 12 weeks at five hours per person per week, my nominal labor goal was 240 hours total per team. The calendar spacing of the milestones was intended to force the students to distribute this labor uniformly across the span of the project.

Table 6: *Summary of Team UCP Estimates, Reported Effort, and PF*

Student Team Project Data	Student Teams						Mean All Teams
	A	B	C	D	E	F	
UCP Estimates	89	74	56	275	53	123	111.6
Reported Effort (hours)	297	262	216	170	171	384	249.7
Actual PF (hours/UCP)	3.34	3.52	3.89	0.62	3.22	3.12	2.95

Student UCP Estimates

The UCP estimates prepared by the student teams are summarized in Table 6 along with reported labor hours required to complete the project. Even though all teams were estimating the same project starting with the same initial list of requirements, a wide range of UCP estimates were produced with the largest estimate of 275 UCPs being approximately five times larger than the smallest estimate of 56 UCPs. Reported effort also varied over a wide range from a low of 170 hours to a high of 384 hours. Calculated productivities ranged from 0.62 to 3.89 hours per use case point, which is significantly lower than Karner's nominal value. To determine the source of the discrepancies, we examine the intermediate calculations below.

Analysis of Estimate Discrepancies

Table 7 summarizes the intermediate UCP calculations for all six teams. The subsequent discrepancy analysis examines the UCP calculations and identifies factors that contributed to the observed variations across the student teams. Recommendations for improving the consistency and quality of the estimates are also presented.

UUCW Calculations

The majority of teams identified between nine and 14 distinct use cases though Team B and Team D identified 18 and 32 use cases respectively. The primary source of discrepancy in the number of use cases identified was the consolidation or expansion of related use cases, that is, a use case with multiple scenarios may have been split and counted as multiple distinct uses cases. An example of this would be the *Reserve Room* use case where each of the four types of room reservations is treated as a separate use case. Splitting of such a use case could inflate the UUCW if the complexity of the resulting use cases did not decrease. Team D's estimate is an extreme example of this situation.

Another source of UUCW discrepancy is in the number of actor-product interactions identified. Some teams described additional interactions for a given use case than did other teams due to either an alternate interpretation of the initial project requirements or feature creep, the incorporation of extra features by the developers. For example, one team included a user authentication interchange with every use case. Other teams included confirmation interchanges to verify user inputs. In other instances, a team would split an interaction into multiple interchanges. For example, the hotel manager should be able to set the base room rate on a given date. Some teams documented the two inputs, base rate and date, as a single use case step while other teams split the input of the two values into multiple steps.

It was also clear that on some teams, no effort was made to make the level of use case detail included uniform across the entire use case model. Instead, use cases were assigned to individual team members and prepared as individual assignments with little or no peer review. Since the UUCW is a measure of the number of steps or interactions between an actor and the system, additional interactions can result in a given use case being classified as more complex during the UCP estimation process, inflating the UUCW.

The first step in determining a reasonable UUCW estimate must be *communication with the customer*. The use case model documents the developer's vision of the product's functionality, yet I, acting as the customer, received surprisingly few questions from the students regarding clarification of specific project requirements. Students developed their products in a vacuum, by choice, and as a result the teams were not really estimating the exact same product due to their respective interpretations of the initial requirements and occasional addition of *nice-to-have* features that were not explicitly required.

A *use case preparation standard* could also improve the UUCW estimation procedure. Such a standard should include specific criteria for combining or splitting of use cases for estimation purposes. Examples must be included in the standard to illustrate the desired level of detail to include in the model for estimation purposes. This level of detail is likely to vary with the scope of the product under consideration and is an area of research that I am currently pursuing.

One should also *perform a reality check on the initial UUCW estimate* derived from the use case model. Karner's UCP method allows you to determine the complexity of a use case from its text description, but as shown in Table 1, he also specified for each complexity category the number of objects required for implementation [1]. As a reality check, one can revisit the initial complexity assessment of each use case by examining the number of objects required in its implementation. If the two assessments disagree, you will have to make some decisions as to how to proceed. The brief summary of Karner's estimation technique presented in [1] provides no guidance on how to deal with this discrepancy. One approach would be to complete the calculation under the worst-case assumption, always sorting use cases into the most complex category identified by either of the two methods. One could also average the complexity weights indicated for those use cases in which the two methods produced different complexity assessments. You could also compare your UUCW estimate to that for any similar products developed by your organization to see if it seems reasonable.

"Some students felt it was easiest to rate a product for a particular factor if it happened to fall at one of the extremes, irrelevant, essential, or unknown, but they expressed a need for some criteria to differentiate the intermediate influence levels."

UAW Calculations

Two types of errors were observed in the student UAW calculations: incorrect identification of the set of actors, and duplicate counting of actors in the calculation. In my view, the system had two actors, a hotel employee and a hotel manager, who directly interacted with the reservation system, and a Guest, who interacted indirectly with the system via the hotel employee or manager. Team B argued that the guest could be omitted for estimation purposes since the guest did not directly interact with the reservation system. Team C included all three actors in their estimate. Team A identified a single actor but classified that actor as complex since they implemented a graphical user interface. While inexperience at use case modeling contributed to these discrepancies, additional guidance was needed on the inclusion or exclusion of indirect actors in the estimate. Teams D-F, however, counted the same actor multiple times for each use case in which that actor was involved, inflating their resulting UAW. Several commercial and freeware UCP estimation software tools have reduced the possibility of this particular error by the way their developers have chosen to guide users through the UCP estimation process.

TCF and ECF Calculations

Both the TCF and ECF calculations evaluate to approximately 1.0 if the influence of all of their corresponding factors are unknown. Recall that the influence values are integers that range from 0 (irrelevant) to 5 (essential) with an influence value of 3 used to indicate that the impact of a particular technical or environmental factor is unknown. Some students felt it was easiest to rate a product for a particular factor if it happened to fall at one of the *extremes*, *irrelevant*, *essential*, or *unknown*, but they expressed a need for some criteria to differentiate the intermediate influence levels.

Fortunately, the TCF proved to be relatively insensitive to variations on the perceived impact of a single technical factor since the weights associated with each factor are small. The computed TCFs were all within 0.08 of the overall mean value 0.79. Some variation in ECF is to be

Table 7: *Summary of Supporting UCP Estimate Calculations*

Metric	Student Team Estimates						Mean All Teams
	A	B	C	D	E	F	
Total Use Cases	14	18	12	32	9	14	16.5
UUCW	105	120	70	310	90	160	142.5
Total Actors	1	2	3	10	9	14	6.5
UAW	3	6	6	18	18	28	13.17
TCF	0.78	0.71	0.80	0.87	0.72	0.87	0.79
ECF	1.06	0.83	0.92	0.97	0.68	0.76	0.87

expected since the ECF gauges people-related factors such as object-oriented development experience, application domain experience, motivation, etc.

PF Calculations

The average productivity, excluding Team D, was approximately 3.5 hours per use case point, which was significantly lower than Karner's nominal value of 20 hours per use case point. Karner's nominal value was determined from real-world products, which are likely to be significantly larger and more robust than those developed by the student teams (see reported labor in Table 5). Given the time constraints of the semester, the student products are essentially prototypes that lack adequate error handling, and in some cases, omit functionality identified in their use case model.

It is also important to remember that these PFs are derived from the labor hours reported by the student teams. Inaccurate reporting of labor hours on student projects does occur. In some cases, students forget to record their hours for the week and report an estimated labor value for that week. Students may also choose to inflate their reported hours in hopes of achieving a better grade on the assignment.

Mathematical Errors

While the mathematical computations required by the UCP estimation method are not conceptually difficult, errors did occur. The best way to minimize such errors is to *provide a software tool* that implements the UCP calculations correctly. While a spreadsheet estimation template would suffice, a dedicated UCP estimation tool that interfaced with your modeling tool would help prevent errors in execution of the estimation method itself, such as duplicate counting of actors in the UAW calculation, in addition to preventing the purely numeric errors.

Conclusions

The student team estimation data illustrates both the potential of the UCPs estimation method and some of the pitfalls that can be avoided. Excluding Teams D-F due to calculation errors, team productivity averaged approximately 3.5 hours per use case point on this semester-sized, four-person project. While this was significantly lower than that reported by Karner [1], the time constraints of the semester prevent the students from developing product-quality code.

Since the PF is determined experimentally, it is important to have a consistent method of producing a use case points estimate. The consistency of the presented estimates could have been improved by *better communication with the customer*, to prevent inclusion of unnecessary features, and by the use of a *use case preparation standard*. Such a standard must address the level of detail to be included in the description of each use case and include specific criteria for splitting or combining use cases. Commercial UCP *software tools* also help to prevent errors by automating many of the UCP calculations. One must also perform a *reality check* on any UCP estimate, using either the required objects count of the UCP method, a best-case/worst-case analysis, or your own historical data, to determine if the method has produced a reasonable estimate. Finally, one must keep in mind that the UCP estimation method is the focus of ongoing research worldwide so your estimation procedures should be reviewed periodically to incorporate the latest lessons learned.◆

References

1. Karner, Gustav. "Resource Estimation for Objectory Projects." Objective Systems SF AB, September 17, 1993.
2. Clemmons, Roy K. "Project Estimation With Use Case Points." CrossTalk Feb. 2006.
3. Schach, Steven R. Object-Oriented & Classical Software Engineering. 6th ed. Boston: McGraw Hill, 2005.

Coming Events

April 1-3
FOSE 2008
Washington, D.C.
www.fose.com

April 7-11
RSA Conference 2008
San Francisco, CA
www.rsaconference.com/2008/US/home.aspx

April 14-18
DoD Enterprise Architecture
Orlando, FL
www.afei.org/brochure/8a05/index.cfm

April 15-17
MRO Military 2008 Conference and Exhibition
Fort Lauderdale, FL
www.aviationweek.com conferences/milmain.htm

April 15-17
Software Test and Performance Conference
San Mateo, CA
www.stpcon.com

April 28-May 1
SATURN 2008
Pittsburgh, PA
www.sei.cmu.edu/architecture/saturn/2008

April 29-May 2
STC Systems & Software Technology Conference
2008 Systems and Software Technology Conference
Las Vegas, NV
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.

About the Author

David J. Coe, Ph.D., is an assistant professor in the Department of Electrical and Computer Engineering at the University of Alabama in Huntsville. He currently teaches undergraduate and graduate courses in C++ programming, data structures, and software engineering, and he has consulted locally in the areas of software engineering and software process. Coe has an undergraduate degree in computer science from Duke University, a masters of science in electrical engineering, and a doctorate in electrical engineering from the Georgia Institute of Technology.

The University of Alabama, Huntsville
Department of Electrical and Computer Engineering
217-F Engineering BLDG
Huntsville, AL 35899
Phone: (256) 824-3583
E-mail: coe@ece.uah.edu

Good Practices for Developing User Requirements

Ellen Gottesdiener
EBG Consulting

Defining user requirements – the needs of the stakeholders who directly interact with the system – is arguably one of the most difficult challenges in building complex systems. When it comes to defining user requirements for software, it is essential to use models to document and analyze the requirements. This article provides a requirements model roadmap that helps software development teams understand the effective use of requirements models. It also describes good practices for creating and using these models.

Many software developers have a love-hate relationship with requirements. They love having a list of things they need to engineer into the product they are building, but they hate it when the requirements are unclear, inaccurate, self-contradictory, or incomplete. They are right to be concerned.

The price is high for teams that fail to define requirements or that do it poorly. Ill-defined requirements result in requirements defects, and the consequences of these defects are ugly [1- 6]:

- Expensive rework and cost overruns.
- A poor quality product.
- Late delivery.
- Dissatisfied customers.
- Exhausted and demoralized team members.

To reduce the risk of software project failure and the costs associated with defective requirements, project teams must address requirements early in software development and they must define requirements properly.

A Short Review of Requirements

Before we get to the nitty-gritty of building requirements models, let us look at some basic requirements concepts. User requirements – the focus of this article – are one of three types of requirements (see Figure 1). The other two types are those related to the mission or business and those that describe the software itself.

Business requirements are statements of the business rationale for the project. These requirements grow out of the vision for the product which, in turn, is driven by mission (or business) goals and objectives. The product's vision statement articulates a long-term view of what the product will accomplish for its users. It should include a statement of scope to clarify which capabilities are and are not to be provided by the product.

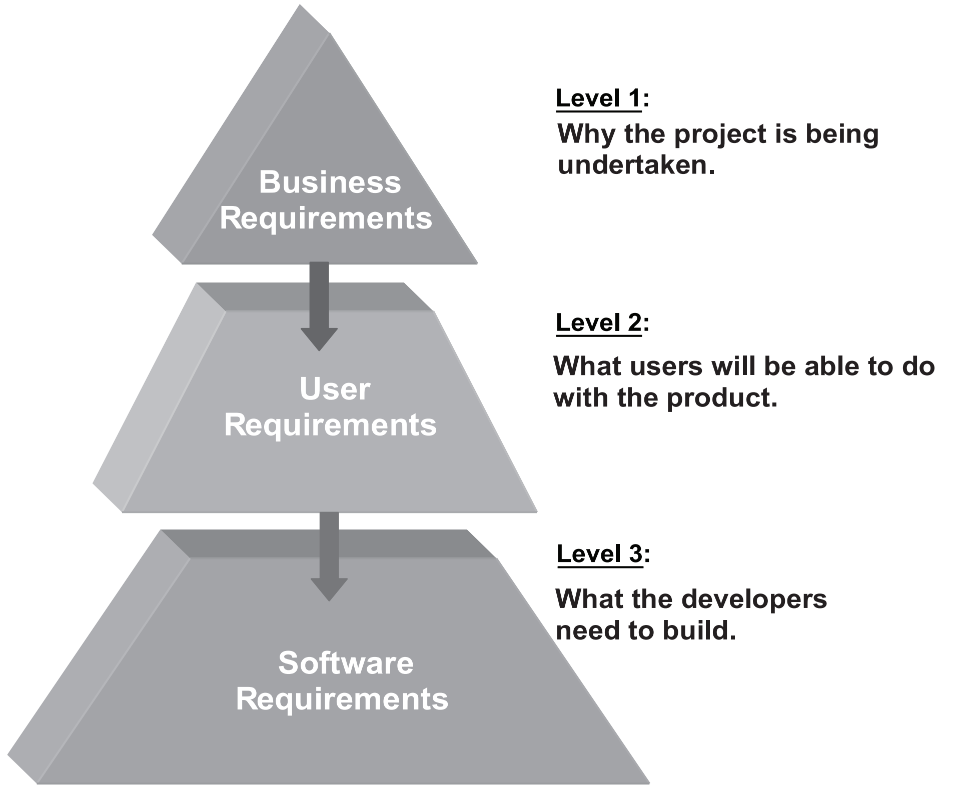
User requirements define the software requirements from the user's point of view, describing the tasks users need to accomplish with the product and the quality requirements of the software from the user's point of view. *Users* can be broadly defined to include not only the people who access the system but also inanimate *users* such as hardware devices, databases, and other systems. In the systems produced by most government organizations, user requirements are articulated in their concept of operations document.

Software requirements are detailed descriptions of all the functional and nonfunctional requirements the software must fulfill to meet business and user needs. Nonfunctional requirements include software design constraints, external interfaces, and quality attributes such as performance, security, installation ability, availability, safety, reusability, and more [7]. Software requirements, which are documented in a software requirements specification, establish an agreement between technical specialists and business managers on what the product must do.

The key activities in requirements development are the following: *elicitation*, *analysis*, *specification*, and *validation* [8]. In *elicitation*, you identify the sources of requirements and solicit requirements from those sources. Requirements elicitation relies on appropriate stakeholder involvement, one of the most critical elements for project success [9]. The goal of requirements *analysis* is to sufficiently understand and define the requirements so that stakeholders can prioritize and allocate them to software. *Specification* involves differentiating and documenting functional and nonfunctional requirements and checking that the requirements are documented unambiguously and completely. *Validation* examines the requirements to ensure that they satisfy user's needs.

Elicitation and analysis are crucial early activities that require intense stakeholder involvement. To analyze the requirements

Figure 1: *Requirements Levels*


Level 1:
Why the project is being undertaken.
Business Requirements
Level 2:
What users will be able to do with the product.
User Requirements
Level 3:
What the developers need to build.
Software Requirements

you are eliciting, a key good practice is to create *requirements models* (also referred to as *analysis models*): user requirements represented by text (such as tables, lists, or matrices), diagrams, or a combination of text and graphical material [7]. These models facilitate communications about requirements with your stakeholders.

As you elicit requirements from stakeholders and represent them using requirements models, you should verify your models to ensure they are internally consistent. You also need to prioritize your requirements: With active user involvement, you analyze the trade-offs among requirements to establish their relative importance [8].

The User Requirements Model Roadmap

Now let us take a closer look at user requirements models. The beauty of the Requirements Model Road Map (Figure 2) is that it shows the relationships between the three types of requirements (business, user, and software) and categorizes the models you can use to represent each type. Each model is designed to answer one of the *5Ws + 1H* questions: *Who? What? When? Why? How?* [7].

(Note that the question *Where?* provides information mainly about nonfunctional requirements. Although these are not user requirements – which depict functional requirements – analysts asking *Where?* during analysis will also discover a slice of useful quality attributes such as performance and usability).

In addition, the user requirements model falls into three categories: scope, high-level and detailed, and alternative models. Some models (shown in italics in Figure 2) are useful for analyzing the business process, and others are useful for clarifying project scope. Defining stakeholder categories early in elicitation, for example, identifies the people you should involve in requirements modeling. High-level and detailed models, such as use cases, a data model, and business rules, can reveal requirements defects such as errors, omissions, and conflicts. Requirements analysts and engineers can substitute alternative models when the engineers better communicate requirements or fit the project culture.

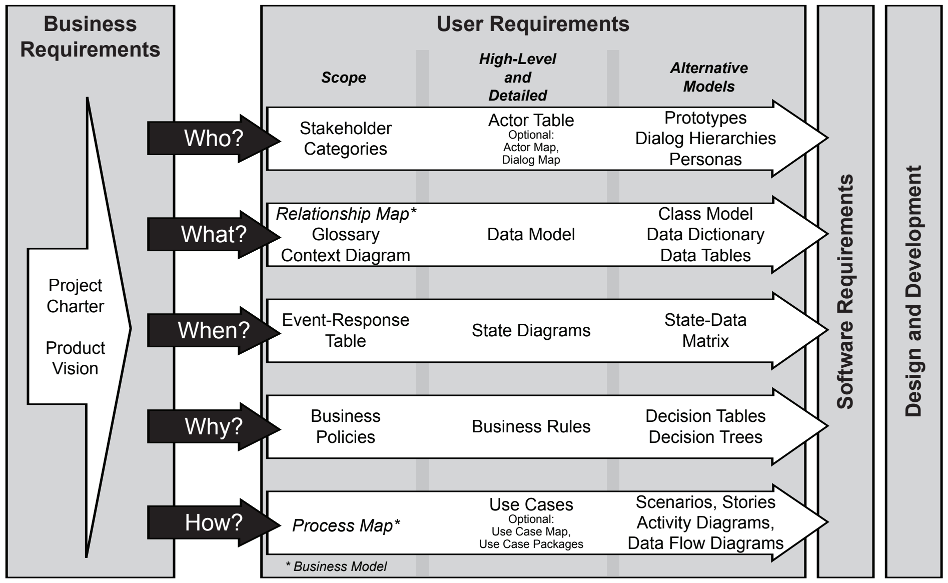
Each requirements model represents information at a different level of abstraction. A model such as a state diagram represents information at a high level of abstraction, whereas detailed textual requirements represent a low level of abstraction. By stepping back from the trees (textual requirements) to look at the forest (a state diagram), the team can discover requirements defects not evident when reviewing textual requirements alone.

Because the requirements models are related, developing one model often leads to deriving another. Examples of one model driving another model are the following:

- Actors initiate use cases.
- Scenarios exemplify instances of use cases.
- A use case acts upon data depicted in the data model.
- A use case is governed by business rules.
- Events trigger use cases.

In this way, you can use various routes to harvest one model from another. This approach helps you develop the models quickly while at the same time verifying the model's completeness and correctness.

Figure 2: *User Requirements Model Roadmap*



User Requirements Models

Here, in alphabetical order, are brief descriptions of the common user requirements models shown in the User Requirements Models Road Map.

Activity Diagram

An activity diagram is a model that illustrates the flow of complex use cases using Unified Modeling Language (UML) notation. This model is useful for showing use case steps that have multiple extension steps, and for visualizing use cases.

Actor Map

An actor map is a model that shows actor interrelationships. An actor map supplements the actor table and can also be used as a starting point for identifying use cases. Actors can be written on index cards (one per index card) or drawn using the UML notation. UML depicts actors in an actor map as stick figures, as boxes (supplemented by the notation "<<Actor>>"), or as a combination (e.g., stick figures for human actors, and boxes for nonhuman actors).

Actor Table

An actor table is a model that identifies and classifies system users in terms of their roles and responsibilities. This model helps reveal missing system users and identifies functional requirements as user goals (use cases), and also management to clarify job responsibilities.

Business Policies

Business policies are guidelines, standards, and regulations that guide or constrain the conduct of a business. Policies are the basis for the decision making and knowledge that are implemented in the software and in manual processes. Whether imposed by an outside agency or from within the company, policies are used to streamline operations, increase customer satisfaction and loyalty, reduce risk, improve revenue, and adhere to legal requirements. This model helps you identify policies allocated to business people, which in turn allows management to prepare for software implementation by updating procedures, guidelines, training, forms, and other assets needed to enforce the policies. Some policies are also allocated to software for implementation.

Business Rules

Business rules are text statements that decompose business policies. Business rules describe what defines, constrains, or enables the software behavior. You use business rules to specify the controls that govern user requirements and to clarify which rules should be enforced in software and which will be allocated to business people. Because business rules require data, defining the rules will uncover needed data. User requirements depend on the complete and correct enforcement of business rules.

Class Model

A class model is a diagram that shows the classes to be used in a system. A *class* is the generic definition of a collection of similar objects (persons, places, events, and physical artifacts). You use a class model in projects employing object-oriented software development methods, tools, or databases.

Context Diagram

A context diagram is a model that shows the system in its environment with the external entities (people and systems) that provide and receive information or materials to and from the system. This model helps stakeholders to quickly and simply define the project's scope and to focus on what the system needs as inputs and provides as outputs. A context diagram helps the team derive requirements models (such as actors, use cases, and data model information) and can reveal possible scope creep problems as new external entities are added.

Data Dictionary

A data dictionary is a model that provides a description of the data attributes and structures used in a system. This model is a central place for defining each data element and describing its data type, length, and format. Some project teams use data modeling tools that provide data dictionary capabilities.

Data Flow Diagram (DFD)

A DFD is a model that shows related inputs, processes, and outputs for processes that respond to external or temporal events. Unlike use cases (which are oriented toward actor goals), DFDs focus on the data that goes in and out of each process, taking an internal view of how the system handles events.

Data Model

A data model shows the informational needs of a system by illustrating the logical structure of data independent of the data design or data storage mechanism. You use a data model to identify, summarize, and formalize the data attributes and structures needed to satisfy functional requirements and to create an easy-to-maintain database. Data models help to simplify design and programming and help identify external data entities (other systems that supply data to the software).

Data Table

A data table is a model in the form of a table that contains sample data to elicit and validate a data model or data dictionary. Each row represents a set of occurrences in an entity, and each column represents sample attributes.

Decision Table

A decision table is a model that specifies complex business rules concisely in an easy-to-read tabular format. A decision table documents all the possible conditions and actions that need to be accounted for in business rules. *Conditions* are factors, data attributes, or sets of attributes and are equivalent to the left side of atomic business rules. *Actions* are conclusions, decisions, or tasks and are equivalent to the right side of atomic business rules. Factors that must be evaluated form the top rows of the table. Actions make up the bottom rows of the table.

Decision Tree

A graphical alternative to a decision table, a decision tree presents conditions and actions in sequence. Each condition is graphed with a decision symbol representing *yes* or *no* (or a *true* or *false* conclusion). Branches to additional conditions are drawn left to right. Actions are drawn inside rectangles to the right of the branch to which they apply.

Dialog Hierarchy

A dialog hierarchy is a model that shows the dialogs in a system (or Web page) as a hierarchy. It does not show transitions.

Dialog Map

A dialog map is a diagram that illustrates the architecture of a system's user interface. It shows the visual elements that users manipulate to step through tasks when interacting with the system. Dialog maps can be used to uncover missing or erroneous use case paths and to validate use cases, scenarios, or both in requirements walkthroughs with users.

Event-Response Table

An event-response table model identifies each event (an input stimulus that triggers the system to carry out a function) and the event responses resulting from those functions. An event-response table defines the conditions to which the system must respond, thereby defining the functional requirements at a scope level. (Each event requires a predictable response from the system.) This model can also reveal needs for external database access or file feeds.

Glossary

The glossary is a list of definitions of business terms and concepts relevant to the software being developed or enhanced.

Persona

The persona is a description of an actor as a fictitious system user or archetype. You describe each persona as if he or she is a real person with personality, family, work background, preferences, behavior patterns, and personal attitudes. The focus is on behavior patterns rather than job descriptions. Each persona description is written as a narrative flow of the person's day with added details about personality. Four or five personas represent the roles that use the system most often or are most important to the functional requirements.

Process Map

A process map is a diagram that shows the sequence of steps, inputs, and outputs needed to handle a business process across multiple functions, organizations, or job roles. This model helps you identify the processes that are allocated to the business (manual processes) and those that will be allocated to software.

Prototype

A prototype is a partial or preliminary version of a system created to explore or validate requirements. Exploratory prototypes can be mock-ups using paper, whiteboards, or software tools.

Relationship Map

A relationship map is a diagram that shows the information and products that are exchanged among external customers, suppliers, and key functions in the organization. This model helps you understand the organizational context of the project by identifying affected business functions and their inputs and outputs.

Scenario

A scenario is a description of a specific occurrence of a path through a use case (i.e., a use case instance). Example: A customer calls to reschedule a job, adding another service and requesting a repeat customer discount.

Stakeholder Categories

Stakeholder categories are structured arrangements of groups or individuals who have a vested interest in the product being developed. You use this model to understand who has an interest in or who has influence on the product, who will use the software and its outputs, and who the product will affect in some way. These groups and individuals will need to be kept informed about requirements progress, conflicts, changes, and priorities.

State-Data Matrix

A state-data matrix model shows attributes that are added or changed during a state change. Each attribute is identified in the data model and data dictionary.

State Diagram

A state diagram is a visual representation of the life cycle of a data entity. Events trigger changes in data, resulting in a new state for that entity. Each state is a defined condition of an entity, a hardware component, or the entire system that requires data, rules, and actions. A state diagram can also show actions that occur in response to state changes. You use a state diagram to understand how events affect data and to identify missing requirements such as events, business rules, data attributes, and use case steps.

Story

A story is a text description of a path through a use case that users typically document. Stories replace use cases and scenarios when you are planning releases for change-driven software projects. Stories are essentially detailed scenarios, but each story is judged by developers to require less than two weeks to develop. When combined with acceptance tests, stories are roughly equivalent to use cases.

Use Case

The use case describes in abstract terms how actors use the system to accomplish goals. Each use case is a logical piece of user functionality that can be initiated by an actor and described from the actor's point of view in a technology-neutral manner. Use cases summarize a set of related scenarios. The purpose of use cases is to reveal functional requirements by clarifying what users need to accomplish when interacting with the system. Use cases are a natural way to organize functional requirements and can be easier for users to understand and verify than textual functional requirements statements.

Use Case Map

The use case map is a model that illustrates the work flow of use cases. Each use case map represents a set of highly cohesive use cases sharing the same data, often triggered by the same events or initiated by the same actor.

Use Case Package

The use case package is a logical, cohesive group of use cases that represents higher level system functionality. You create a use case package by combining use case maps or grouping use cases. Most systems will have multiple packages. You can use a UML file folder notation to show each package, and you can name each package according to its functionality.

Good Practices for Modeling User Requirements

Following good requirements modeling practices (see *Good Practices for Modeling User Requirements*, Table 1) is the key to successful development of user requirements. These practices accelerate modeling, engage stakeholders, and give you high-quality requirements – ones that are correct, complete, clear, consistent, and relevant.

The first good practice is to represent and agree on the project's scope early in requirements elicitation. Why? It has to do with scope creep – the unrestrained expansion of requirements as the project proceeds. Scope creep is one of the greatest risks in software development [6]. A clear definition of product scope narrows the project's focus to enable better planning, better use of time, and better use of resources. Moreover, scope-level models establish a common language that team members can use to communicate about the requirements and help to articulate the boundary between what is in and what is not in scope for the product.

Another good practice, as mentioned earlier, is to document your product using multiple user requirements models. Each model describes one aspect of a problem the product will address. Thus, no single model can describe all the requirements. Furthermore, elements of one model often link to elements of another, so one model can be used to uncover related or missing elements in another model.

It is also good to use both text and graphics to represent user needs. Multiple representations tap into different modes of human thinking. Some people think more precisely with words, and others understand concepts more quickly via diagrams. Using both types of representations leverages these different thinking modes. In addition, mixing text and graphics makes requirements development more interesting and engaging. It provides variety and permits stakeholders to understand their requirements from more than one angle.

You should also select models that fit the domain of your product. That is because some models are better suited to communicate requirements for certain domains. For example, *When* models (such as an event-response table and a state machine diagram) are well suited to dynamic domains – those that respond to continually changing events to store data and act on it based on its state at a point.

Another well-known good practice is to develop your requirements iteratively. Each iteration is a self-contained mini-project in which you undertake a set of activities – elicitation, analysis, specification, and validation – resulting in a subset of requirements. The rationale for this practice is that user requirements seldom remain unchanged for a long period. On teams using agile methods, each iteration also incorporates the work needed to deliver the working software that satisfies those requirements. In some domains, requirements change faster than the system or subsystem can be developed. In addition, the cost of implementing changes increases dramatically as the project proceeds. Developing requirements in an evolving manner is essential in reducing these risks.

You can also use requirements models to identify requirements defects. The interconnections among the models help to expose any inconsistencies in related models. This self-checking accelerates the team's ability to uncover missing, erroneous, vague, or conflicting requirements.

Table 1: *Summary: Good Practices for Modeling User Requirements*

1.	Define, represent, and agree on the project's scope early in requirements elicitation.
2.	Document your product using multiple user requirements models.
3.	Select models that fit the domain of your system.
4.	Develop requirements models iteratively.
5.	Use requirements models to identify requirements defects.
6.	Use models to communicate: Create simple, readable diagrams focused less on beauty and more on understanding.
7.	Conduct retrospectives as you iterate through requirements development.

When you are creating graphical models, it is crucial to create simple, readable diagrams. The benefit of diagrams is that they give you a way to quickly communicate complex, controversial, or unclear requirements. Thus, you should avoid complex, hard-to-read diagrams. Draw diagrams manually to begin with or use an easy-to-learn drawing tool. Keep them simple and easy to read. Focus on maintaining accuracy and exposing unclear or incorrect requirements – not beauty or completeness.

The final good practice I want to mention applies whether or not you are using modeling: I always tell my clients to conduct short retrospectives at the end of each requirements iteration. A *retrospective* is a special meeting in which the team explores what works, what does not work, what can be learned from the just completed iteration, and what ways to adapt their processes and techniques before starting another iteration [10, 11]. Retrospectives allow for early learning and correction and may be your team's most powerful tool for process improvement.

On Your Way

Software development teams enjoy access to a world of tools and technologies, but building truly successful software still depends on team members gaining a deep understanding of user needs. When your team is developing a software product, you will save time, money, and frustration by using appropriate models to describe and analyze the product's user requirements.◆

References

1. Reifer, Donald J. "Profiles of Level 5 CMMI Organizations." CrossTalk Jan. 2007.
2. Schwaber, Carey. "The Root of the Problem: Poor Requirements." IT View Research Document. Forrester Research, 2006
3. Dabney, James B., and Gary Barber. "Direct Return on Investment of Software Independent Verification and Validation: Methodology and Initial Case Studies." Assurance Technology Symposium, June 2003 <http://sarpresults.ivv.nasa.gov/ViewResearch/24.jsp>.
4. Hooks, Ivy F., and Kristina A. Farry. Customer-Centered Products: Creating Successful Products Through Smart Requirements Management. New York: Amacom, 2001.
5. Nelson, Mike, James Clark, and Martha Ann Spurlock. "Curing the Software Requirements and Cost Estimating Blues." The Defense Acquisition University Program Manager Magazine Nov.-Dec. 1999.
6. Jones, Capers. Patterns of Software Systems Failure and Success. Boston, MA: Thomson Computer Press, 1996.
7. Gottesdiener, Ellen. Software Requirements Memory Jogger: A Pocket Guide to Help Software and Business Teams Develop and Manage Requirements. Methuen, MA: Goal/QPC, 2005.
8. Institute of Electrical & Electronics Engineers (IEEE). "IEEE Software Engineering Body of Knowledge." IEEE Computer Society, 2004 <www.swebok.org>.
9. Standish Group International. "CHAOS Chronicles." Standish Group International, 2003.
10. Kerth, Norman L. "Project Retrospectives: A Handbook for Team Reviews." New York: Dorset House, 2001.
11. Gottesdiener, Ellen. "Team Retrospectives for Better Iteration Assessment." The Rational Edge Apr. 2003 <http://ebgconsulting.com/Pubs/Articles/TeamRetrospectives-Gottesdiener.pdf>.

About the Author

Ellen Gottesdiener, principal consultant, EBG Consulting, helps get the right requirements so projects start smart and deliver the right product at the right time. Her book, "Requirements by Collaboration: Workshops for Defining Needs" describes how to use multiple models to elicit requirements in collaborative workshops, and "The Software Requirements Memory Jogger" describes essentials for requirements development and management. In addition to providing training, eLearning and consulting services, she speaks at and advises for industry conferences, writes articles, and serves on the Expert Review Board of the International Institute of Business Analysis Business Analysis Body of Knowledge.

EBG Consulting, Inc.
1424 Ironwood DR West
Carmel, IN 46033
Phone: (317) 844-3747
E-mail: ellen@ebgconsulting.com

Tabular Notations for State Machine-Based Specifications

Markus Herrmannsdörfer, Dr. Sascha Konrad, and Brian Berenbach
Siemens Corporate Research

Finite state machines are a widely used concept for specifying the behavior of reactive systems. Numerous graphical notations based on finite state machines have been developed and are commonly used today, such as state transition diagrams, Harel statecharts, and Unified Modeling Language (UML) state machine diagrams. While not as widely used, tabular notations for state machine-based specifications offer complementary advantages to diagrammatic notations. In this article, we describe five approaches using tabular notations for state machine-based specifications and evaluate these approaches for use in software development.

The term *reactive system* describes a system that needs to continuously react to inputs coming from the environment. Finite state machines are a widely used concept for specifying the behavior of such systems. Since finite state machines allow the rigorous capture of functional aspects of system behavior[1], they offer several advantages over informal specifications. For example, they provide the ability to automatically generate code or test cases, and they enable formal verification and validation (V&V). Generally, a finite state machine is an appropriate representation when a problem or solution has the following characteristics:

- Finite and discrete set of states (e.g., on, off, and standby).
- Discrete and manageable set of inputs.
- Change of state is only performed in response to an input (e.g., if a button is pressed, then the machine transitions from state off to state on).

State machines[2] are used for specifying functional properties for a wide variety of systems, such as control systems and user interfaces. For example, Siemens uses state machines to precisely specify the circuitry in mail sorting systems and the controls in car radios. They are also the paradigm of choice for software compiler design and programmatic interpretation of natural language. Numerous graphical notations for state machines have been developed and are commonly used today, such as state transition diagrams, Harel statecharts [1], and UML state machine diagrams [2]. Graphical notations are often preferred by developers, analysts, and testers over textual information, since diagrams allow the visualization of complex relationships.

Tabular notations for state machines (commonly also referred to as *state tables* or *state transition tables*) offer complementary advantages to these graphical notations. For example, the incompleteness of a specification, i.e., the actions of the system in a specific state in response to a specific event that are not addressed by the specification, can easily be identified as empty cells in the table. In addition, tabular notations are relatively compact and have shown to scale well to practical systems [3]. Due to these reasons, tabular notations for state machines are preferred in some domains over graphical notations for the rigorous specification of system behavior. For instance, Siemens Automotive commonly receives system requirements in the form of state tables, captured in either Excel sheets or proprietary databases.

While a tabular representation is relatively compact and the completeness of the requirements specification can easily be determined, it has been shown to cause numerous difficulties. For instance, the requirements specification for a system of realistic size is often quite large and of considerable complexity, consisting of numerous large tables. As a result, precisely understanding the required behavior solely through visual inspection is difficult. Moreover, requirements captured in simple Excel sheets are difficult to analyze for consistency and adherence to critical properties.

This article presents and evaluates several state machine-based tabular notations that can address some of the aforementioned problems. For instance, some notations enhance the understandability of the specification by offering a complementary graphical representation. In addition, hierarchical composition is used by several notations to keep the specification tractable and some provide tool support for V&V. The remainder of this article is organized as follows: the Background section provides an overview of finite state machines and Harel statecharts. The Tabular Notations for State Machines section describes five approaches using tabular notations for state machine-based specifications. We conclude by evaluating these notations for use in software development with respect to several factors.

Background

This section introduces finite state machines, including a common graphical and tabular notation, and briefly describes the advanced features of Harel statecharts.

Finite State Machine

The term *finite state machine* describes a class of computational models that consist of a finite set of states, a start state, a set of inputs (events), and a transition function that determines the next state of the finite state machine based on the current state and input [4]. The finite state machine starts computation in the start state; transitions between states are performed based on the transition function. Numerous variants of this basic type of state machine exist. For example, Moore machines extend finite state machines with outputs (actions) associated with states, while Mealy machines associate outputs with transitions [5]. For the remainder of this article, we use Mealy machines as the computational model. Finite state machines may be deterministic or non-deterministic. In deterministic finite state machines for a given input, one transition can be taken from the current state, at most. In non-deterministic finite state machines, however, one input may enable several transitions of which one is then taken.

A common way of representing finite state machines is the use of *state transition diagrams* (commonly also referred to as *state diagrams*). State transition diagrams are directed graphs in which states are depicted as nodes and transitions are represented by directed edges. Transitions are commonly labeled with the triggering events and actions, using the following general syntax: *trigger/action(s)*. Figure 1 contains a sample state transition diagram showing the simple behavior of a door: The door can be opened or closed.

If the door is closed, then it can be locked or unlocked. The door can only be opened when it is unlocked. If the door is closed (irrespective of being locked or unlocked), then it can be pushed in. Because of this event, an alarm will sound and the door will then be permanently in the state *Broken*.

In addition to the graphical representation, finite state machines may be specified using state transition tables. A state transition table denotes the action performed by the automaton and the next state based on the current state (row) and event that occurred (column). A dash denotes that no such transition exists. A state transition table representing the automaton specified in Figure 1 can be seen in Table 1. Using this tabular notation, completeness of the specification can be readily established. Since a cell needs to be labeled explicitly with a dash if no such transition exists, a cell that does not contain a destination state or a dash renders the specification incomplete. Using a graphical notation, determining the completeness of the specification is more difficult, since a missing arrow in a diagram could potentially be the result of an omission, but could also mean that no such transition exists.

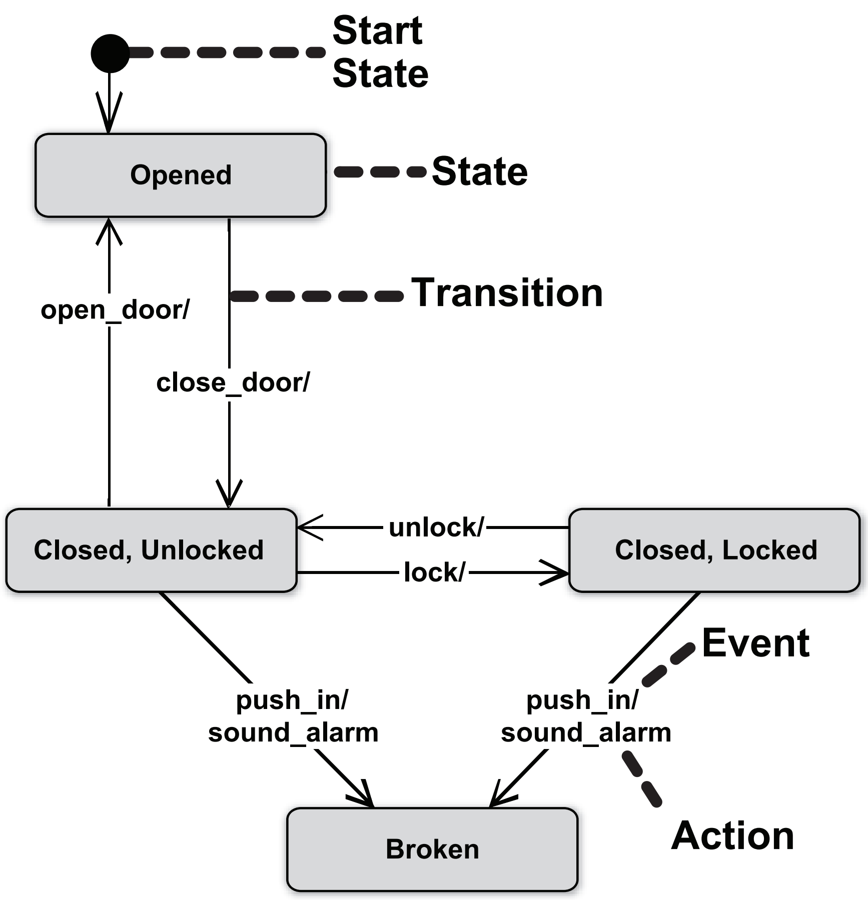


Figure 1: *Sample State Transition Diagram*

Harel Statecharts

While finite state machines have shown to be useful for modeling reactive systems, their representation as state transition diagrams does not scale well to large-scale systems and may become *unstructured*, *unrealistic*, and *chaotic*. To address this problem, David Harel developed statecharts that extend state transition diagrams with the following concepts [1]:

1. **Depth.** Commonly also referred to as *XOR (eXclusive OR) decomposition* or *state nesting*. Using state nesting, a state may be a composite state that contains exactly one region serving as a container for sub states. To be in the composite state, the system must be in exactly one of its sub states (which itself may be composite states again).
2. **Orthogonality.** Also known as AND decomposition. Using *AND decomposition*, a state may be a composite state comprising two or more orthogonal regions executing independently and concurrently. Therefore, to be in the composite state, the system must be in a state of all of its orthogonal regions at the same time. Each orthogonal region may itself contain additional composite states.
3. **Broadcast communication.** Since orthogonal regions are independent and execute concurrently, the computational model of statecharts uses broadcast communication. As a result, each orthogonal region receives occurring events and may take transitions that had become enabled.

In addition to these basic extensions, Harel statecharts provide additional constructs such as entry and exit actions for states, conditionals, and history states (see [1] for more details). The UML notation for state machines is based on Harel statecharts and uses a number of these extensions. (For a detailed comparison of the syntax and semantics of UML state machine diagrams and Harel statecharts, please refer to [6].) Figure 2 (see page 20) shows how statecharts provide more structure and reduce the perceived complexity of the diagrammatic representation of the door example in comparison to the state transition diagram in Figure 1. For instance, after the introduction of a composite state *Closed* in Figure 2, describing the behavior of the door when it is pushed in requires only one transition, which makes the diagram appear cleaner and less cluttered. In general, the extensions provided by Harel statecharts and UML state machine diagrams have shown to be an effective means to reduce the perceived complexity of state machine representations for reactive systems. For example, the authors in [7] performed some studies with university students and concluded that the use of composite states in UML state machine diagrams improves understandability.

Table 1: *State Transition Table Corresponding to Figure 1*

Event / State	open_door	close_door	lock	unlock	push_in
Opened	-	/Closed, Unlocked	-	-	-
Closed, Unlocked	/Opened	-	/Closed, Locked	-	sound_alarm/ Broken
Closed, Locked	-	-	-	/Closed, Unlocked	sound_alarm/ Broken
Broken	-	-	-	-	-

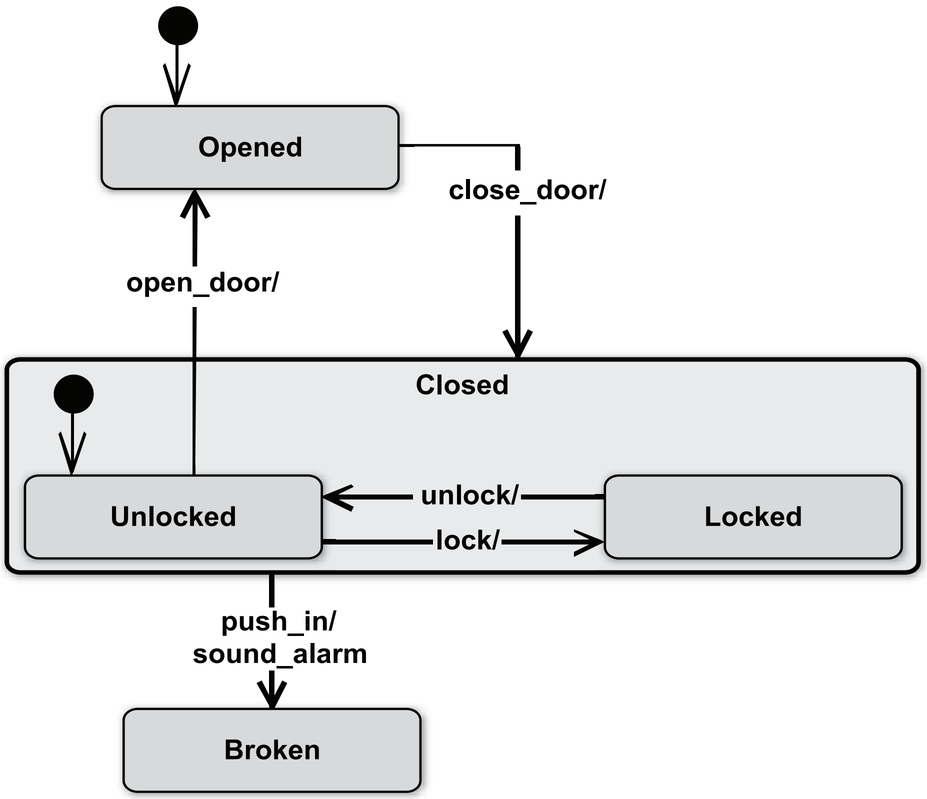


Figure 2: *Sample Statechart*

Tabular Notations for State Machines

This section describes five approaches that use tabular notations for state machine-based models, namely Virtual Finite State Machines (VFSM) [8], Software through Pictures (StP) [9], Parnas Tables [10], Software Cost Reduction (SCR) [3], and the Requirements State Machine Language (RSML) [11].

VFSMs

The VFSM is a concept for the specification of control systems in a virtual environment. The environment is termed virtual since events and actions of the environment are represented by abstract names for inputs and outputs in the state machine [8]. The behavior of the system may be specified as a *state table* that shows actions and transitions performed in a certain state based on specific conditions. Table 2 shows a sample state table for *State₁*. Upon entering the state, *Output₁* is always produced and upon exiting the state, *Output₂* is always produced. If *Condition₁* is satisfied, then *Output₃* is produced without causing a state change (internal transition). However, if *Condition₂* is satisfied, then *Output₄* is produced and the state machine transitions to *State₂* (external transition).

While VFSMs support entry and exit actions, they do not support state nesting or orthogonality. However, different sets of concurrent high-level and low-level finite state machines can be created and connected to achieve structuring through hierarchical decomposition [12].

The application of VFSMs is facilitated by StateWORKS Studio, a tool suite for creating specifications using VFSMs [13]. The tool suite offers an editor that combines and synchronizes diagrammatic and tabular views of VFSMs. In addition, a simulator and an executor are provided that can be used to validate and execute VFSM specifications.

StP

StP Structured Environment (SE) is a tool-supported approach for specifying a system using diagrammatic notation complemented with tabular notation [9]. The behavior of a system is specified in terms of control flow diagrams and state transition diagrams. Complementary to state transition diagrams, two tabular notations are provided: *state event matrix* and *state transition table*.

A state event matrix shows all transitions of the state machine in a grid of source states and triggering events. Similar to the state transition table shown in Table 1, a transition is entered into the cell at the intersection of its source state (row) and its triggering event (column). The cell contains the list of actions to perform and the target state of the transition. Table 3 shows an example state event matrix, in which the state machine transitions from *State₁* to *State₂* upon occurrence of *Event₁*, producing *Action₁*, and it transitions back to *State₁* upon occurrence of *Event₂*, producing *Action₂*. If *Event₃* occurs in *State₁*, then the state machine performs *Action₃* but remains in the current state.

A state transition table shows all transitions of a state machine in a list (refer to Table 4). The tabular layout provides a column for the source state, the triggering event, the action, and the target state.

Table 2: *VFSM State Table*

State name	Condition(s)	Action(s)	
$State_1$	Entry action	$Output_1$	
	Exit action	$Output_2$	
	$Condition_1$	$Output_3$	Internal transitions
	…	…	
$State_2$	$Condition_2$	$Output_4$	External transitions
	…	…	

Table 3: *StP SE State Event Matrix*

State	Event		
	$Event_1$	$Event_2$	$Event_3$
$State_1$	$\frac{Action_1}{State_2}$		$\frac{Action_3}{State_1}$
$State_2$		$\frac{Action_2}{State_1}$	

The tabular notations provided by StP SE are compact and readable. However, diagrams and tables of StP SE provide neither state nesting nor orthogonal regions. Similar to VFSMs, structuring is possible using hierarchical decomposition. In order to facilitate the implementation phase, the StP SE tool suite provides code generation and reverse engineering capabilities for the C programming language.

Current State	Event	Action	Next State
$State_1$	$Event_1$	$Action_1$	$State_2$
$State_1$	$Event_3$	$Action_3$	$State_1$
$State_2$	$Event_2$	$Action_2$	$State_1$

Table 4: *StP SE State Transition Table*

Parnas Tables

Parnas and Madey developed the four-variable model as an underlying state machine model to formally specify system requirements [14]. The name of the model arises from the fact that a specification contains four distinct sets of variables:

- Variables monitored by the system (MON).
- Variables controlled by the system (CON).
- Variables that the input devices of the system read from (INPUT).
- Variables that the output devices of the system write to (OUTPUT).

The relations between the variable sets of the four-variable model are illustrated in Figure 3.

Specifically, the variables are linked by the following five relations:

- Natural constraints on the monitored and controlled variables (NAT).
- Expected change of controlled variables in response to changes in monitored variables, i.e., the actual system requirements (REQ).
- Relation of monitored variables to input variables (IN).
- Relation of controlled variables to output variables (OUT).
- Relation between input and output variables, realized by software (SOFT).

A possible notation for expressing these relations are Parnas Tables [10]. Parnas Tables are a collection of 10 table types for capturing functional and relational expressions, each having a distinct syntax and semantics. A developer should choose the table format that produces a simple, compact representation for expressing the relation at hand. For each table type, rules exist for identifying incompleteness and inconsistency.

Table 5 (see page 22) contains a sample Parnas Table of type decision table. A *decision table* can represent a function or relation where the domain is an ordered set of potentially distinct types. One dimension of the table itemizes the elements of the domain. Table 5 shows the syntax of a decision table representing the relation between two variables, A and B, and a decision that is made based on the values of these variables. For instance, Table 5 states that **if** $A = A_2$ **and** $B = B_2$ **then make** $Decision_2$.

Parnas Tables do not support nesting or orthogonality, but allow the developer to reference a function that is defined in a different table. Since Parnas Tables have completely formal semantics, tool support can be developed to check the tables automatically. However, to the best of our knowledge, such tool support is not currently available.

SCR

SCR is a set of formal methods for the design of software systems [3]. Similar to Parnas tables, SCR also uses the four-variable model as its underlying abstraction, and the relationships between monitored and controlled variables are captured in tables [10, 14].

In order to capture the relations concisely, SCR defines modes. A mode describes a set of system states in which the system exhibits equivalent behavior in response to events and conditions. Mode classes then describe the relationships between these modes and are modeled in terms of mode transition tables. In order to model complex systems with independent components, several mode classes may be constructed to capture concurrency. The occurrence of an event is denoted by a value change of a condition. $@T$ is used to specify that a condition becomes *true*, while $@F$ specifies that a condition becomes *false*.

SCR uses three different types of tables to specify a system: *condition tables*, *event tables,* and *mode transition tables*.

A condition table defines the value of a variable depending on the mode and a condition. For example, in Table 6, the variable var_3 is assigned the value *greater*, *equal*, or *less* in the modes M_1 and M_2 of mode class MC_1, depending on the values of variables var_1 and var_2.

An event table defines the value of a variable as a function of the mode and a (possibly conditioned) event. For example, Table 7 (see page 22) assigns variable var_4 a *true* or *false* value in the modes M_1 and M_2 of mode class MC_1 based on a change in value of the variable var_3.

Finally, the mode transition table defines how the mode of a mode class changes in response to events. A sample mode transition table for the mode class MC_1 is given in Table 8. Specifically, the system changes from mode M_1 to mode M_2 upon variable var_4 becoming *true*, and switches back to mode M_1 if the variable becomes *false*. Commonly, a mode transition table contains only events that change the mode; events that do not cause mode changes are omitted to increase readability.

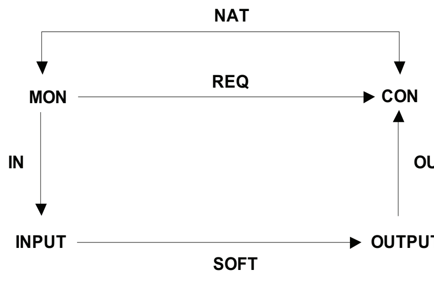
The SCR notation is rigorous and compact, but purely tabular. Nesting and orthogonality are not supported by the notation, but hierarchical decomposition can be used to structure complex systems. Tools to support various V&V approaches have been developed [3]. Once the system model is complete, the model can be checked for different types of errors, such as incompleteness or ambiguities. In addition, a simulator can be used to run scenarios and inspect whether the results are as expected.

RSML

The RSML was originally developed to write requirements specifications for process-control systems such as a collision avoidance system for a commercial airliner [11]. RSML combines a graphical notation based on Harel statecharts with a tabular notation for specifying transition conditions. As such, RSML retains most of the advanced features of statecharts, such as depth and orthogonality, while using tables to facilitate the readability of conditions associated with transitions.

RSML specifications generally consist of state diagrams with unlabeled transitions. Transitions are not labeled in order to increase the readability of a state diagram when enabling conditions of transitions are complex. Instead of using labels,

Figure 3: *Four-Variable Model* [14]



	Decision$_1$	Decision$_2$	Decision$_3$
A	A_1	A_2	A_3
B	B_1	B_2	B_3

Table 5: *Parnas Decision Table*

Mode Class MC_1	Conditions		
M_1, M_2	$var_1 < var_2$	$var_1 = var_2$	$var_1 > var_2$
var_3	greater	equal	less

Table 6: *SCR Condition Table for Variable Var₃*

Mode Class MC_1	Events	
M_1, M_2	@T(var_3 = equal)	@T(var_3 = greater) OR @T(var_3 = less)
var_4	true	false

Table 7: *SCR Event Table for Variable Var₄*

Old Mode	Event	New Mode
M_1	@T(var_4)	M_2
M_2	@F(var_4)	M_1

Table 8: *SCR Mode Transition Table for Mode Class MC₁*

properties of transitions are defined separately from the diagrams in transition definitions. A transition definition contains the source and destination of the transition, the state machine where the transition is located, the triggering event, the guarding condition, and the output action. The guarding condition of a transition is defined in terms of AND/OR tables. A sample AND/OR table can be seen in Table 9, which describes that the associated transition is enabled (after the triggering event has occurred) when *Expression₁* is *true* AND *Expression₂* is *false* at the same time, OR *Expression₃* is found to be *true*. The period denotes that the truth value of the expression is irrelevant for the current evaluation.

The final RSML specification can then be checked for consistency and completeness. In addition, techniques

Table 9: *RSML AND/OR Table*

OR

A	Expression$_1$	T	·
N	Expression$_2$	F	·
D	Expression$_3$	·	T

have been developed that allow the analysis of RSML specifications using theorem proving and model checking techniques [15]. As such, the correctness of an RSML specification can be rigorously established. Similar to SCR, tools supporting the simulation of RSML specifications also exist.

Conclusions

Due to advanced syntactical and semantical features, Harel statecharts and UML state machine diagrams are better suited than the basic state transition diagram notation to handle complex systems. Similarly, the five presented approaches use advanced features that make them better suited for complex systems than basic state transition tables. Analysts or developers that need to determine which approach to use in their development processes should consider several factors. If the main goal is to facilitate the implementation phase and reduce coding efforts, then the *VFSMs* and *StP* approaches seem preferable, since they offer commercially available tool support that allows executing or generating code from the specifications.

However, if the focus is the formal analysis of the system specification for various completeness and correctness properties, Parnas Tables, SCR, and RSML are better suited. Since Parnas Tables do not offer tool support and require the user to understand the syntax and semantics of each possible table type, they can only be recommended to developers with a solid understanding of formal methods that do not need automated support for formal analysis. In contrast to Parnas Tables, SCR and RSML offer mature tool support for V&V. When deciding between SCR and RSML, an important factor may be the availability of a graphical representation. While SCR is purely tabular, RSML uses the tabular representation only for capturing the guarding conditions of transitions, while states and unlabeled transitions are captured in terms of diagrams. We believe that such a combination of diagrammatic and tabular views combines the specific advantages each of these views offer. In addition to combining graphical and tabular views, RSML also supports the concepts of nesting and orthogonality of Harel statecharts. These concepts have shown to be effective means to reduce the perceived complexity of models. Dutertre and Stavridou have examined the use of SCR and RSML for an avionic storage management system specification and concluded that RSML specifications are commonly easier to understand than SCR specifications due to these structuring features [16].

In conclusion, we believe that while tabular notations for state machines are not a *silver bullet* solution, they may greatly facilitate the specification and analysis of systems in specific domains. Tabular notations seem to be explicitly useful for systems with a large number of transitions between states or rather complex enabling conditions of transitions. In order to retain the advantage of having a graphical view, the presented VFSM, StP SE, and RSML approaches use tables and diagrams. As such, they attempt to combine the complementary advantages of diagrammatic and tabular notations. In addition, if the system under design is rather complex (i.e., having a large number of states and transitions), notations supporting nesting and orthogonality may provide a significant reduction in specification complexity. The mature V&V tool support offered by some of the presented approaches also offers significant advantages over specifications using Excel tables or proprietary databases, where often the only available means of analysis is visual inspection.◆

Notes

1. Non-functional aspects, such as performance and reliability, are usually captured by different means.
2. For the remainder of this article, we assume that the system being specified has a finite set of states and we use the terms finite state machine and state machine interchangeably.

References

1. Harel, D. "Statecharts: A Visual Formalism for Complex Systems." Science of Computer Programming 8.3 (1987).
2. Object Management Group. "UML 2.0 Superstructure Specification." 2004 <www.omg.org/cgi-bin/doc?formal/05-07-04>.
3. Heitmeyer, D. "Tools for Constructing Requirements Specifications: The SCR Toolset at the Age of Ten." International Journal of Computer Systems Science and Engineering 20.1 (2005) <http://chacs.nrl.navy.mil/publications/CHACS/2005/2005heitmeyer-finalJCSSE.pdf>.
4. National Institute of Standards and Technology (NIST). "Finite State Machine." Dictionary of Algorithms and Data Structures. NIST 2006 <www.nist.gov/dads/HTML/finiteStateMachine.html>.
5. Hopcroft, J., and J.D. Ullman. Introduction to Automata Theory, Language, and Computation. Reading, MA: Addison-Wesley, 1979.
6. Crane, M., and J. Dingel. "UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal." Lecture Notes in Computer Science 2005 <www.cs.queensu.ca/~stl/papers/MoDELS2005.pdf>.
7. Cruz-Lemus, J.A., M. Genero, M. Esperanza Manso, and M. Piattini. "Evaluating the Effect of Composite States on the Understandability of UML Statechart Diagrams." Lecture Notes in Computer Science 3713 (2005) <www.giro.infor.uva.es/Publications/2005/CGMP05/CruzLemusGeneroMansoPiattini-Models05.pdf>.
8. Wagner, F. VFSM Executable Specification. Proc. of the International Conference on Computer Systems and Software Engineering. The Hague, Netherlands, 1992 <www.stateworks.com/active/download/wagf92-software-engineer ing.pdf>.
9. Aonix. "Software Through Pictures." 2006 <www.aonix.com/stp.html>.
10. Parnas, D.L. "Tabular Representation of Relations." CRL Report 260. Ontario, Canada: McMaster University, 1992.
11. Leveson, N., M.P. Heimdahl, H. Hildreth, and J. Reese. "Requirements Specification for Process-Control Systems." IEEE Transactions on Software Engineering 20.9 (1994) <sunnyday.mit.edu/papers/tcas-tse.pdf>.
12. Wagner, F., and P. Wolstenholme. "Modeling and Building Reliable, Re-Useable Software." Workshop on Model-Based Development of Computer Based Systems. Huntsville, AL, 2003 <www.stateworks.com/active/download/wagf03-2-modeling-reliable-software.pdf>.
13. StateWORKS Software. "StateWORKS Studio." 2006 <www.stateworks.com>.
14. Parnas, D.L., and J. Madey. "Functional Documentation for Computer Systems Engineering." Science of Computer Programming 25.1 (1995).
15. Choi, Y., and M.P. Heimdahl. "Model Checking RSML-e Requirements." Proc. of the 7th IEEE international Symposium on High Assurance Systems Engineering (HASE '02). 2002 <www.umsec.umn.edu/files/47.73.model-checking-rsml.pdf>.
16. Dutertre, B., and V. Stavridou. "Avionics Systems Requirements: A Comparison of RSML and SCR." Proc. of the 16th International System Safety Conference, Seattle, WA, 2002 <www.csl.sri.com/papers/issc98/issc98.ps>.

About the Authors

Markus Herrmannsdörfer is a Ph.D. student at the Chair of Software and System Engineering at Technische Universität Munchen, Germany. During his master-level studies, he worked on the subject of this publication as an intern at Siemens Corporate Research in Princeton, New Jersey. His current focus is on metamodeling, especially on the problem of metamodel evolution.

Technische Universität München
Institut für Informatik
Boltzmannstr. 3
85748 Garching bei München
Germany
Phone: +49 (89) 289-17336
E-mail: herrmama@in.tum.de

Sascha Konrad, Ph.D., is a consultant at Siemens Corporate Research. He received his intermediate diploma from the University of Kaiserslautern, Germany, and his master's and doctorate degrees in computer science and engineering from Michigan State University. Konrad's research interests include requirements engineering and automated analysis of software specifications, including software patterns, the Unified Modeling Language, agile and model-driven software development, formal methods and computer-aided verification, and distributed and embedded systems.

Siemens Corporate Research
755 College RD East
Princeton, NJ 08540
Phone: (609) 734-6500
E-mail: sascha.konrad @siemens.com

Brian Berenbach is the technical program manager for requirements engineering at Siemens Corporate Research. He has been working in the field of requirements engineering for more than 15 years, first as a consultant, and then as a senior member of the technical staff at Siemens. Recently, his program has been involved with requirements definition for such diverse products as medical systems, baggage handling, mail sorting, automated warehouses, and embedded automotive systems.

Siemens Corporate Research
755 College RD East
Princeton, NJ 08540
Phone: (609) 734-6500
E-mail: brian.berenbach @siemens.com

Addressing the Challenges of Tactical Information Management in Net-Centric Systems With DDS

Dr. Douglas C. Schmidt, Dr. Angelo Corsaro, and Hans van't Hag
PrismTech Corporation

Recent trends in net-centric systems motivate the development of tactical information management capabilities that ensure the right information is delivered to the right place at the right time to satisfy quality of service (QoS) requirements in heterogeneous environments. This article presents an architectural overview of the Object Management Group's (OMG) Data Distribution Service (DDS), which is a standards-based QoS-enabled data-centric middleware platform that enables applications to communicate by publishing information they have and subscribing to information they need in a timely manner. DDS is an important distributed software technology for mission-critical Department of Defense (DoD) net-centric systems because it supports the following: (1) location independence, via anonymous publish/subscribe (pub/sub) protocols that enable communication between collocated or remote publishers and subscribers, (2) scalability, by supporting large numbers of topics, data readers, and data writers and platform portability, and (3) interoperability, via standard interfaces and transport protocols.

Tactical information management systems increasingly run in net-centric environments characterized by thousands of platforms, sensors, decision nodes, and computers connected together to exchange information, support sense-making, enable collaborative decision making, and effect changes in the physical environment. For example, the Global Information Grid (GIG) is an ambitious net-centric environment being designed to ensure that different services and coalition partners, as well as individuals participating to specific missions, can collaborate effectively and deliver appropriate firepower, information, or other essential assets to warfighters in a timely, dependable, and secure manner [1]. Achieving this vision requires the following capabilities from the distributed middleware software:

- **Shared operational picture.** A key requirement for mission-critical net-centric systems is the ability to share an operational picture with planners, warfighters, and operators in real-time.
- **Ensure the right data gets to the right place at the right time** by satisfying end-to-end QoS requirements, such as latency, jitter, throughput, dependability, and scalability.
- **Interoperability and portability in heterogeneous environments.** Since net-centric systems are faced with unprecedented challenges in terms of platform and network heterogeneity, they are necessary.
- **Support for dynamic coalitions.** In many net-centric tactical information management systems, dynamically formed coalition of nodes will need to share a common operational picture and exchange data seamlessly.

Prior middleware technologies (such as the Common Object Request Broker Architecture [CORBA] Event Service and Notification Service, the Java Message Service [JMS], and various other proprietary middleware products) have historically lacked key architectural and QoS capabilities, such as dependability, survivability, scalability, determinism, security, and confidentiality needed by net-centric systems for tactical information management. To address these limitations – and to better support tactical information management in net-centric systems like the GIG – the OMG has adopted the DDS specification, which is a standard for QoS-enabled data-centric pub/sub communication aimed at net-centric tactical information management systems [2]. DDS is used in a wide range of military and commercial systems including naval combat management systems, commercial air traffic control, transportation management, automated stock trading systems, and semiconductor fabrication devices.

The remainder of this article presents an overview of DDS that is geared to software architects. We also discuss the DDS QoS policies that are the most relevant for net-centric tactical information management systems. Finally, we explain how DDS has been applied in practice to address key challenges of developing and operating distributed software in current and planned net-centric tactical information management systems.

Overview of DDS

DDS provides the following capabilities for net-centric tactical information systems:

- **Universal access to information** from a wide variety of sources that run over potentially heterogeneous hardware/software platforms and networks.
- **An orchestrated information bus** that aggregates, filters, and prioritizes the delivery of this information to work effectively under the restrictions of transient and enduring resource constraints.
- **Continuous adaptation to changes** in the operating environment, such as dynamic network topologies, publisher and subscriber membership changes, and intermittent connectivity.
- **Standard QoS policies and mechanisms** that enable applications and administrators to customize the way information is delivered, received, and processed in the appropriate form and level of detail to users at multiple levels in net-centric systems.

This section describes the key capabilities and entities in DDS and shows how its QoS policies can be used to specify and enforce performance-related requirements of net-centric tactical information management systems. Figure 1 shows the various profiles and layers in the DDS standard. The lower layer defines a Data-Centric Publish Subscribe (DCPS) platform, whose goal is to provide efficient, scalable, predictable, and resource-aware data distribution. The higher layer is the Data Local Reconstruction Layer (DLRL), which is an optional interface that provides an object-oriented facáde atop the DCPS. The DLRL can be used to map topics onto object fields and defines navigable associations between objects.

A separate specification, called the Real-Time Publish/Subscribe (RTPS) DDS interoperability wire protocol, defines the standard network protocol used to exchange data between publishers and subscribers that use different implementations of DDS [3]. The remainder of this section describes the conceptual model of DDS and explains the QoS policies that are most relevant for net-centric tactical information management systems.

DDS Conceptual Model

Domains and Partitions

DDS applications send and receive data within a domain. Domains can be divided into partitions that allow the separation and protection of different data flows. Although DDS entities can belong to different domains, only participants within the same domain can communicate, which helps isolate and optimize communication within communities that share common interests. For example, each communication layer within the GIG could be associated with a DDS domain and further subdivided into partitions. This approach isolates domain participants across layers, which enables effective use of resources and helps enforce security and confidentiality policies.

Global Data Space

DDS provides a strongly typed global data space within each domain in which applications produce and consume the dynamically changing portions of a shared information model, as shown in Figure 2. DDS' information model capabilities are similar to those of relational databases, except that DDS' global data space is completely distributed, QoS-aware, and allows anonymous and asynchronous sharing of a common information model. The DDS information model is the only knowledge publishers and subscribers need to communicate, i.e., they need not be aware of each other nor be concerned with low-level network programming details, such as Internet protocol addresses, port numbers, remote object references, or service names. By allowing data to flow where and when needed, DDS's global data space enables the sharing of tactical information and situational awareness information needed to implement net-centric tactical information management systems.

Topic

A DDS topic is an association between a data type, a set of QoS, and a unique name, as shown in Figure 3 (see page 26). A topic is also the unit of information contained in DDS' global data space and is used by applications to define their information model and associate QoS policies with it. DDS applications in net-centric systems define their information model by identifying topics that are relevant for solving their requirements and organizing them into either relational or object-oriented models. DDS thus allows the expression of the system information model as either a 1) topic relational model, which can be thought of as an extension of the familiar entity relationship diagrams used in data bases, decorated with QoS, or 2) an object-oriented model, which can also be synthesized as an object-oriented view of the relational model.

The DCPS layer provides support for relational modeling, while the DLRL extends the DCPS with an object-oriented facade, so that applications can either completely ignore the DCPS relational models or build an object model atop the DLRL. Data associated with DDS topics are expressed using types defined by the standard OMG Interface Definition Language (IDL), which simplifies the inter-working between DDS and CORBA. Relationships between topics can be captured via keys that can be used to distinguish between different instances of the same topic.

In net-centric tactical information systems, an information model will be associated with every layer in which DDS-based data exchange occurs. This information model, which can comply with DoD or North American Trade Organization standards, is the lingua franca used by the different applications in coalitions to exchange information and seamlessly interoperate. Likewise, the QoS policies decorating the information model determine how the data is disseminated, persisted, and received in the global data space.

Publishers and Subscribers

In net-centric tactical information management systems, publishers and subscribers correspond to a range of domain participants such as embedded devices, Unmanned Air Vehicles (UAVs), soldiers' equipment, as well as planning and simulation services in operations centers. DDS applications use data writers to publish

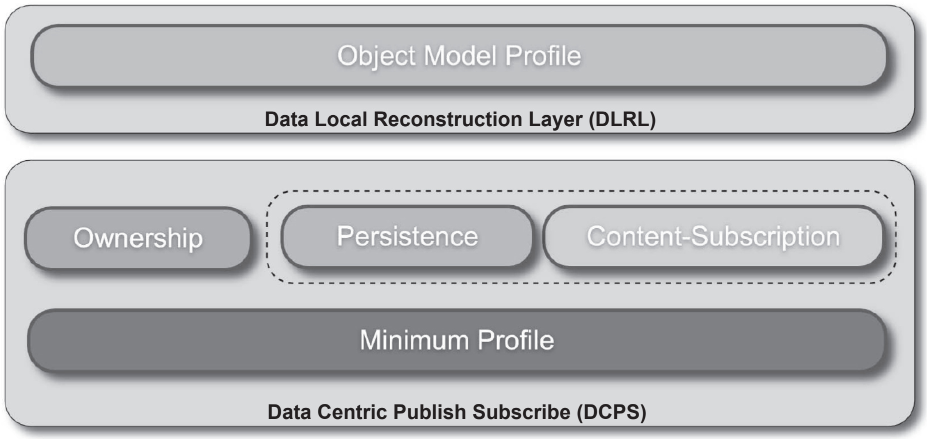
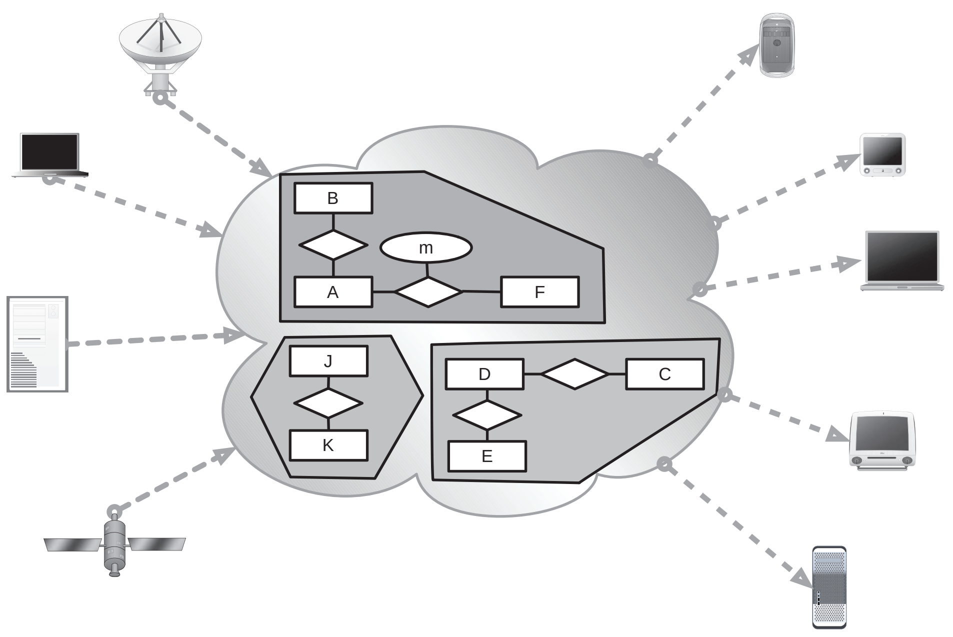

Object Model Profile
Data Local Reconstruction Layer (DLRL)
Ownership
Persistence
Content-Subscription
Minimum Profile
Data Centric Publish Subscribe (DCPS)

Figure 1: *Profiles and Layers in the DDS Standard*

Figure 2: *DDS Global Data Space in a Domain*



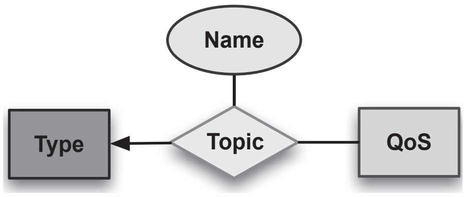


Figure 3: *DDS Topic*

data values to the global data space of a domain and data readers to receive data. A publisher is a factory that creates and manages a group of data writers with similar behavior or QoS policies, as shown in Figure 4. A subscriber is a factory that creates and manages data readers, as shown in Figure 4.

Publishers can declare their intent to produce data on particular topics with associated QoS, and they distribute the data on those topics to the global data space. Subscribers receive topic data in the global data space that match their subscriptions (the rules that define what represents a matching subscription are described below). QoS policies allow publishers and subscribers to define, first, their local behavior, such as the number of historical data samples they require and the maximum update-rate at which they want to receive data, and, second, how data should be treated once in transit with respect to reliability, urgency, importance, and durability. Topics can also be annotated with these QoS policies to drive the behavior of the data-distribution. The QoS policies of pre-defined topics serve as defaults for publishers and subscribers and can therefore ensure consistency between requested and offered QoS.

Subscriptions and Matching

A subscription is an operation that associates a subscriber to its matching publishers, as shown in the center of Figure 4. In addition to the topic-based subscriptions described, DDS also supports *content-based subscription*, in which a subset of the standard Structured Query Language (SQL) is used to specify subscription filters. In DDS a *matching* subscription must match the following two types of a topic's properties: (1) its features, such as its type, name, key, and content; (2) its QoS policies, which are described in the *QoS Policies* section.

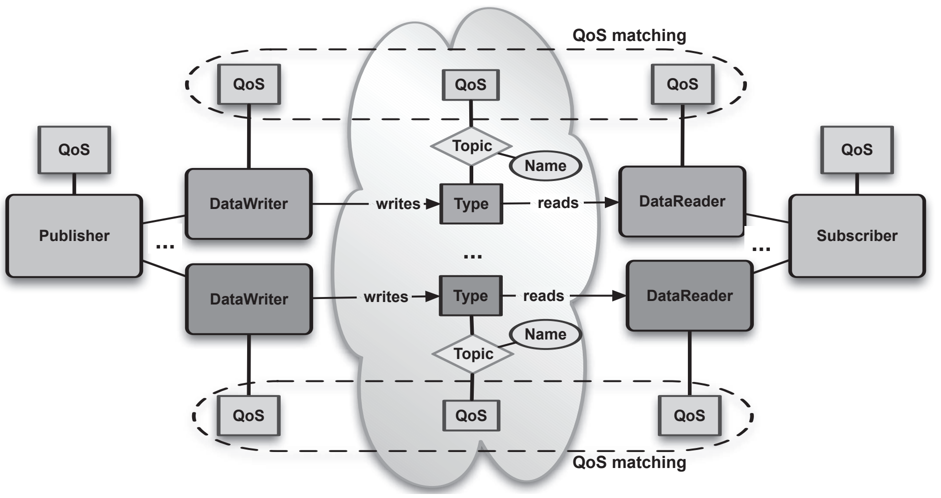
The matching process for QoS uses a requested/offered (RxO) model shown in Table 1, where the requested QoS must be less than or equal to the offered QoS. For example, subscribers requesting reliable data delivery cannot communicate with publishers that only distribute data using best effort delivery. Likewise, subscribers cannot request a topic update whose deadline is smaller than that declared by any publishers.

The subscription matching mechanism provided by DDS enforces a powerful form of *design by contract* [4], where QoS is used together with type information to decide whether publishers and subscribers can communicate. This extended form of design by contract helps ensure that net-centric systems will operate as intended, both from functional and QoS perspectives. These assurances are essential in the development, deployment, and operation of mission-critical net-centric tactical information management systems.

Discovery

Another key feature of DDS is that all information needed to establish communication can be discovered automatically, in a completely distributed manner. Applications dynamically declare their intent to become publishers and/or subscribers of one or more topics to the DDS middleware, which uses this information to establish the proper communication paths between discovered entities. This capability supports dynamic scenarios common in net-centric tactical information management where cooperating domain participants join and leave throughout system operation.

Figure 4: *DDS Publisher/Writer Subscriber/Reader and Subscription/QoS Matching*



QoS Policies

DDS is designed for mission-critical net-centric systems where the right answer delivered too late becomes the wrong answer. To meet timing requirements it is essential that the middleware controls and optimizes the use of resources, such as network bandwidth, memory, and CPU time. Table 1 shows the rich set of QoS policies that DDS provides to control and limit topic (T), data reader (DR), data writer (DW), publisher (P), and subscriber (S) resources and topic QoS properties, such as persistence, reliability, and timeliness [2]. Below we discuss the DDS QoS policies that are the most relevant for net-centric tactical information management systems.

Data Availability

DDS provides the following QoS policies that control the availability of data to domain participants:

- The Durability QoS policy controls the lifetime of the data written to the global data space in a domain. Supported durability levels include the following: (1) volatile, which specifies that once data is published it is not maintained by DDS for delivery to late joining applications; (2) transient local, which specifies that publishers store data locally so that late joining subscribers get the last published item if a publisher is still alive; (3) transient, which ensures that the global data space maintains the information outside the local scope of any publishers for use by late joining subscribers; and (4) persistent, which ensures that the global data space stores the information persistently so it is available to late joiners even after the shutdown and restart of the system. Durability is achieved by relying on a durability service whose properties are configured by means of the DURABILITY_SERVICE QoS.
- The LIFESPAN QoS policy controls the interval of time during which a data sample is valid. The default value is infinite, with alternative values being the time-span for which the data can be considered valid.
- The HISTORY QoS policy controls the number of data samples (i.e., subsequent writes of the same topic) that must be stored for readers or writers. Possible values are the last sample, the

QoS Policy	Applicability	RxO	Modifiable	
DURABILITY	T, DR, DW	Y	N	Data Availability
DURABILITY SERVICE	T, DW	N	N	
LIFESPAN	T, DW	-	Y	
HISTORY	T, DR, DW	N	N	
PRESENTATION	P, S	Y	N	Data Delivery
RELIABILITY	T, DR, DW	Y	N	
PARTITION	P, S	N	Y	
DESTINATION ORDER	T, DR, DW	Y	N	
OWNERSHIP	T, DR, DW	Y	N	
OWNERSHIP STRENGTH	DW	-	Y	
DEADLINE	T, DR, DW	Y	Y	Data Timeliness
LATENCY BUDGET	T, DR, DW	Y	Y	
TRANSPORT PRIORITY	T, DW	-	Y	
TIME BASED FILTER	DR	-	Y	Resources
RESOURCE LIMITS	T, DR, DW	N	N	
USER_DATA	DP, DR, DW	N	Y	Configuration
TOPIC_DATA	T	N	Y	
GROUP_DATA	P, S	N	Y	

Table 1: *Key QoS Policies for Net-Centric Systems*

last *n* samples, or all samples.

These QoS policies provide the DDS global data space with the ability to cooperate in highly dynamic environments characterized by continuous joining and leaving of publisher/subscribers. This capability makes it possible for net-centric tactical information management systems to share a common operational picture regardless of the dynamism that characterizes portions of the systems, such as coalitions of soldiers collaborating in urban environments or coordinated UAVs in support of tactical operations.

Data Delivery

DDS provides the following QoS policies that control how data is delivered and which publishers are allowed to write a specific topic:

- The PRESENTATION QoS policy gives control on how changes to the information model are presented to subscribers. This QoS gives control on the ordering as well as the coherency of data updates. The scope at which it is applied is defined by the access scope, which can be one of INSTANCE, TOPIC, or GROUP level.
- The RELIABILITY QoS policy controls the level of reliability associated with data diffusion. Possible choices are RELIABLE and BEST_EFFORT distribution.
- The PARTITION QoS policy gives control over the association between DDS partitions (represented by a string name) and a specific instance of a publisher/subscriber.
- The DESTINATION_ORDER QoS policy controls the order of changes made by publishers to some instance of a given topic. DDS allows the ordering of different changes according to source or destination timestamps.
- The OWNERSHIP QoS policy controls which writer owns the write-access to a topic when there are multiple writers and ownership is EXCLUSIVE. Only the writer with the highest OWNERSHIP_STRENGTH can publish the data. If the OWNERSHIP QoS policy value is shared, multiple writers can concurrently update a topic. OWNERSHIP thus helps to manage replicated publishers of the same data.

These QoS policies control the reliability and availability of the data, thus allowing the delivery of the right data to the right place at the right time. More elaborate ways of selecting the right data are offered by the DDS content-awareness profile that allows applications to select information of interest based upon their content.

Data Timeliness

DDS provides the following QoS policies that control the timeliness properties of distributed data:

- The DEADLINE QoS policy allows applications to define the maximum inter-arrival time for data. DDS can be configured to automatically notify applications when deadlines are missed.
- The LATENCY_BUDGET QoS policy provides a means for applications to inform DDS of the urgency associated with transmitted data. The latency budget specifies the time period within which DDS must distribute the information. This time period starts from the moment the data is written by a publisher until it is available in the subscriber's data-cache ready for use by reader(s).
- The TRANSPORT_PRIORITY QoS policy allows applications to control the importance associated with a topic or with a topic instance, thus allowing a DDS implementation to prioritize more important data relative to less important data.

These QoS policies make it possible to ensure that tactical information needed to reconstruct the shared operational picture is delivered in a timely manner.

Resources

DDS defines the following QoS policies to control the network and computing resources that are essential to meet data dissemination requirements:

- The TIME_BASED_FILTER QoS policy allows applications to specify the minimum inter-arrival time between data samples, thereby expressing their capability to consume information at a maximum rate. Samples that are produced at a faster pace are not delivered. This policy helps a DDS implementation optimize network bandwidth, memory, and processing power for subscribers that are connected over limited bandwidth networks or which have limited computing capabilities.
- The RESOURCE_LIMITS QoS poli-

cy allows applications to control the amount of message buffering performed by a DDS implementation.

DDS's QoS policies support the various elements and operating scenarios that constitute net-centric tactical information management. By controlling these QoS policies it is possible to scale DDS from low-end embedded systems connected with narrow and noisy radio links, to high-end servers connected to high-speed fiber-optic networks.

Configuration

The QoS policies described above provide control over the most important aspects of data delivery, availability, timeliness, and resource usage. In addition, DDS also supports the definition and distribution of user specified bootstrapping information via the following QoS policies:

- The USER DATA QoS policy allows applications to associate a sequence of octets to domain participant data readers and data writers. This data is then distributed by means of the DCPS participant built-in topic. This QoS policy is commonly used to distribute security credentials.
- The TOPIC_DATA QoS policy allows applications to associate a sequence of octets with a topic. This bootstrapping information is distributed by means of the DCPS Topic built-in topic. A common use of this QoS policy is to extend topics with additional information, or meta-information, such as eXtensible Markup Language schemas.
- The GROUP_DATA QoS policy allows applications to associate a sequence of octets with publishers and subscribers. This bootstrapping information is distributed by means of the DCPS subscription,and DCPS publication built-in topics, respectively. A typical use of this information is to allow additional application control over subscriptions matching.

DDS Success Stories

Although DDS is a relatively new standard (adopted by the OMG in 2004), it has been adopted quickly due to its ability to address key requirements of data distribution in net-centric systems, as well as the maturity and quality of available implementations, which are based on decades of experience developing data-centric middleware for mission-critical systems. Moreover, DDS has been mandated by the U.S. Navy's Open Architecture Computing Environment as the standard publish/subscribe technology to use in next-generation combat management systems, and Defense Information Systems Agency as the standard technology for publish/subscribe to be used in all new or upgraded systems [5, 6]. Several major defense programs, such as the U.S. Navy's DDG-1000 land attack destroyer, U.S. Army's Future Combat Systems (FCS), and the Thales Tactical Information And Command System Operating System (TACTICOS), also adopted DDS even before it was mandated, underscoring DDS' ability to address the data distribution challenges of next generation net-centric defense systems.

For example, the TACTICOS combat management system developed by Thales Naval Netherlands is based on an implementation of DDS that allows them to achieve very good scalability, from small ships to aircraft carrier grade, as well as high performance, availability, and determinism even under temporary overload conditions [7, 8]. TACTICOS is currently in use in 15 navies worldwide serving 20 ships-classes ranging from small patrol boats up to large frigates. The utilization of DDS is instrumental in its success since it provides both the scalability to support thousands of applications running on more than 150 distributed computers on a frigate size system. Another key feature of DDS is its battle-damage resistance, meaning that software can be dynamically re-allocated to the remaining computer pool in case of an error on a specific computer. The DDS Persistence Profile support is instrumental in this dynamic reallocation since it allows applications to store their internal state into the DDS middleware, which manages this state in a distributed and fault-tolerant way so that restarted applications can continue what they were doing before they crashed.

"DDS provides the fault-tolerant information backbone onto which all these applications are deployed and is thus responsible for providing each application with the right information at the right time."

The DDS implementation used on TACTICOS supports a data-centric approach where at the start of the system design, the information model can be captured, annotated with proper QoS policies, and then shared between multiple parties. This federated architecture is common in existing and planned *coalition-based* developments where multiple parties jointly implement the overall combat system. DDS provides the fault-tolerant information backbone onto which all these applications are deployed and is thus responsible for providing each application with the right information at the right time.

Along with the rapid adoption of DDS in the defense domain, its use is also steadily growing in other domains, such as transportation, telecommunications, and finance. For example, in the context of Air Traffic Control and Management, DDS has been selected as the publish/subscribe middleware for distributing flight data plans in CoFlight [9], which is the next generation European Flight Data Processor. In general, DDS is an appropriate middleware technology for application domains that require rich support for QoS policies and high-performance and dependability standards-based, commercial-off-the-shelf implementations.

Concluding Remarks

DDS is a standards-based QoS-enabled data-centric publish/subscribe middleware that provides a feature rich data-centric real-time platform to support the needs of current and planned net-centric tactical information management systems. Its powerful set of QoS policies, together with its scalable architecture, makes it an effective and mature choice for solving the data distribution and information management problems net-centric systems [10]. Next, we summarize how DDS addresses the key challenges outlined in the introduction in a standard and interoperable manner:

- **Shared operational picture.** DDS provides effective support for these types of applications via its QoS policies for defining the scope, content, and QoS of the data model that underlies the operational picture.
- **The right data at the right time at the right place** via DDS QoS policies that enable a fine-grained control over information delivery, such as the ability to control many aspects of data dissemination to ensure timely delivery and optimal resource usage.
- **Heterogeneous environment.** By

providing standard QoS policies that control the bandwidth used for providing data to interested parties, DDS runs in heterogeneous platforms while providing different elements with a common operational picture.

- **Dynamic coalitions.** The highly dynamic nature of DDS, such as its support for dynamic discovery, provides an effective platform for supporting ad hoc interactions.

DDS continues to evolve to meet new operational and technical challenges of net-centric tactical information management systems. Three types of extensions are currently being pursued for DDS by the OMG. The first involves adding new platform-specific models that fully leverage programming language features, such as standard C++ containers. The second extension deals with extensible topics that enable incremental system updates by ensuring that changes in the data model do not break interoperability. The final set of extensions focus on network data representation and the syntax used to define topics. For example, upcoming versions of the DDS standard will likely allow the definition of topics using XML, as well as the use of XML or Java Script Object Notation as the network data representation. DDS security has not yet been standardized. The OMG will be addressing this area of standardization starting in the spring of 2008.

With multiple COTS and open-source implementations and a solid track record of success in mission-critical military and commercial projects, DDS has a bright future as the standards-based middleware of choice for net-centric tactical information systems. More information on DDS and its application in practice are available in online forums [11, 12] where experts discuss advanced features of the DDS standard and new directions for the technology, while DDS beginners can learn from past experiences and ask questions about patterns and best practices for applying DDS in their net-centric systems.◆

References

1. "The United States Department of Defense Quadrennial Defense Review Report." Feb. 2006 <www.defenselink.mil/qdr/report/Report20060203.pdf>.
2. OMG. "Data Distribution Service for Real-Time Systems Specification." <www.omg.org/docs/formal/04-12-02.pdf>.
3. OMG. "Real-Time Publish Subscribe Protocol – DDS Interoperability Wire Protocol Specification." <www.omg.org/cgi-bin/apps/doc?ptc/06-08-02.pdf>.
4. Bertrand, Meyer. Object Oriented Software Construction. 2nd ed., Prentice Hall, 2001.
5. "Open Architecture Computing Environment." <www.nswc.navy.mil/wwwDL/B/OACE>.
6. Defense Systems Information Agency. "DoD Information Technology Standards Registry." <https://disronline.disa.mil>.
7. THALES. "TACTICOS Combat Management System, Exploiting the Full DDS Potential." <www.omg.org/docs/dds/06-12-06.pdf.>.
8. "OpenSplice DDS." <www.prismtech.com/openplice-dds>.
9. CoFlight eFDP <www.omg.org/docs/dds/07-07-04.pdf>.
10. Xiong, Ming, et al. "Evaluating Technologies for Tactical Information Management in Net-Centric Systems." Proceedings of the Defense Transformation and Net-Centric Systems Conference, Apr. 9-13, 2007, Orlando, FL.
11. OMG DDS SIG Portal <portals.omg.org/dds>.
12. OMG DDS Forum <www.dds-forum.org>.

About the Authors

Douglas C. Schmidt, Ph.D., is a professor of computer science at Vanderbilt University and is the chief technical officer of PrismTech. His expertise focuses on distributed computing middleware, object-oriented patterns and frameworks, and distributed real-time and embedded systems. He has authored nine books and more than 350 papers in top technical journals, conferences, and books that cover high-performance communication software systems, real-time distributed computing, and object-oriented patterns for concurrent and distributed systems.

PrismTech Corporation
6 Lincoln Knoll LN
STE 100
Burlington, MA 01803
Phone: (781) 270-1177
Fax: (781) 238-1700
E-mail: doug.schmidt@prismtech.com

Angelo Corsaro, Ph.D., is the OpenSplice DDS product marketing manager at PrismTech and co-chairs the OMG DDS Special Interest Group and the Real-Time Embedded and Specialized Services task force. He is well-known in the distributed real-time and embedded systems middleware community and has a wealth of experience in hard real-time embedded systems, large-scale, and very large-scale distributed systems such as defense, aerospace, homeland security and transportation systems.

PrismTech Corporation
6 Lincoln Knoll LN
STE 100
Burlington, MA 01803
Phone: (781) 270-1177
Fax: (781) 238-1700
E-mail: angelo.corsaro@prismtech.com

Hans Van't Hag is the OpenSplice DDS product manager, at PrismTech. He has extensive experience in applying an information approach towards mission-critical and real-time net-centric systems. Hag is a contributor to the OMG DDS specification and has presented numerous papers on DDS and publish/subscribe middleware technologies. Prior to joining PrismTech, he worked at Thales Naval Netherlands (TNN) where he was responsible for the development of the data-centric real-time middleware as applied in TNN's combat system in service with 15 navies worldwide.

PrismTech Corporation
6 Lincoln Knoll LN STE 100
Burlington, MA 01803
Phone: (781) 270-1177
Fax: (781) 238-1700
E-mail: hans.vanthag@prismtech.com

Systems & Software Technology Conference

29 April – 2 May 2008 • LAS VEGAS, NEVADA

Technology: Tipping the Balance

Welcome to the 20th installment of the Systems and Software Technology Conference (SSTC). Just about 20 years ago, Tetris was becoming the computer game of choice. VGA Graphics were gaining in popularity. A company known by its initials, IBM, was delivering the PS/2 computer. It had a funny new pointing device - a mouse. And the first ever Software Technology Conference was held. From the beginning, SSTC has focused on technology relating to the Department of Defense. We begin another decade with this same focus.

Our theme will be ***"Technology: Tipping the Balance"***. The idea behind the theme is to explore new and needed technologies, as well as lessons learned, which tip the balance in the favor of our Defense Services, providing them an asymmetric advantage.

Submittal Topics Included:

- Assurance and Security
- Estimating and Measuring
- Lessons Learned
- New Concepts and Trends
- Policy and Standards
- Processes and Methods
- Professional Development
- Robust Engineering
- Systems: from Design to Delivery

Who Should Attend:

- Acquisition Professionals
- Program/Project Managers
- Programmers
- System Developers
- Systems Engineers
- Process Engineers
- Quality and Test Engineers

For Complete Conference & Trade Show Information

www.sstc-online.org

REGISTER TODAY!

Marathon or Sprint?

Recently, I had the opportunity to be stuck in an airplane in Albuquerque, heading towards Atlanta. We boarded the plane a few minutes late, but managed to pull out of the gate and head to the runway within 10 minutes of the scheduled time.

Once we got to the runway, we sat. And sat. And sat… Eventually, the plane engines sped up, and everyone breathed a sign of relief as the plane departed. Unfortunately, all it did was move forward a few hundred feet, take a taxiway, and head back towards the terminal. Passengers were immediately asking "What's happening?" The flight attendants never made an announcement and neither did the flight crew. People started calling on their cell phones, initializing the frantic *can-I-get-rebooked-on-another-flight-or-airline?* calls.

We approached the terminal and stopped short. At that time, the pilot finally came on the intercom and said, "The temperature/humidity conditions are marginal, so we've decided to de-ice the plane before takeoff. We'll be off in 15 minutes, and we're predicting that we'll arrive in Atlanta only 5 to 10 minutes late. Everybody should have no problem making their connections." There were massive sighs of relief as we all smiled and realized that there were no catastrophic schedule changes in store for us. Everybody relaxed, except for me – I thought of how this would be a perfect topic for a BACKTALK.

In software development, we also have passengers. We call them customers. And, just like the passengers on the airplane, they feel they have a right to know what is happening.

The problem arises when it comes down to *How Much Do I Tell My Customer*? At what level do you start sharing (infrequent) good and (frequent) bad news?

I have consulted on software projects that have ranged from full and open communication to *just keep telling them we are on schedule*. And, as expected, these projects have also ranged, in a different dimension, from "Wow – delivered on time and on budget!" to "Well, time to update the old resume." And in all cases, open communication made the difference between (on yet another dimension) "Wow, what a great team of developers!" to "Can Tony Soprano arrange a hit for me?"

Here's a secret: *All* software development projects have a lot of politics involved. And, to misquote a famous author[1], "Politics are like sausages – it is best not to watch them being made." Internal politics make for changes, changes result in bad news, and that requires communication between developers and customers.

On the developer side, people come and go. Good developers often leave, retire, get sick, take a better job, burn out, freak out, sneak out…you get the picture. They hire new personnel, retrain others from other projects (projects that have had their budget cut, were cancelled, etc). Critical folks take leave at the worst time. And every time developer staff changes, productivity metrics changes. Earned value moves around

On the customer side, requirements change. Frequently. Not because we didn't know them, but because of politics. The nine different stakeholders can't agree on interfaces. Small issues become huge (and yet, huge issues seldom become trivial – pity).

And through it all, both sides have to develop a cutoff bar of how much information is shared with the other side. It's not about us vs. them, it's about politics. It's about contract issues. It's about budgeting. But, underneath it all, it's about trying to get the project finished on time, on budget, and with happy end users. Delivering bad news is never a good thing – but it's often a necessary thing.

Being a good consultant, I have a workable answer to the question *how much information do I share?* Put yourselves in their shoes, and ask yourself would their job be easier if they knew more about the politics that I am involved in? Can I tell them additional information about upcoming events without doing damage to myself (or my organization)?

You don't want to be one of the pick-up-the-phone-and-tell-them-everything kind. You need a filter mechanism to filter out garbage and a metric to figure out how much to tell them. It probably requires a centralized communication channel – don't undercut the boss by telling the other side things he or she might not want discussed.

The best advice I have ever seen given to a development team was "Remember, it's a marathon, not a sprint[2]." In case you don't know the origin of the word marathon, it comes from the legend of Pheidippides, a Greek soldier, who was sent from the town of Marathon to Athens to announce that the Persians had been defeated in the Battle of Marathon. It is said that he ran the entire distance without stopping and burst into the senate with the words "Masters! Victory is ours!" before collapsing and dying due to exhaustion.

I don't think the messenger (or the project) should die. Marathon was about a victory, not a defeat! When we win the battle, as we cross the finish line, we might be exhausted in victory, but it will be because of the words we have said up to that point, not what we gasp as we finish. *Please* don't kill the messenger.

— David A. Cook
dcook@aegistg.com
The AEgis Technologies Group, Inc.

Notes

1. Nobody really knows who said "Laws are like sausages – it is best not to see them being made." See <http://en.wikiquote.org/wiki/Otto_von_Bismarck>. It has been attributed to Otto von Bismarck, Winston Churchill, Benjamin Disraeli, Clarence Darrow, Mark Twain, and Kaiser Wilhelm, among others.
2. Oddly enough, the development team was using the Agile methodology Scrum, which has periods of development called "Sprints." Come on, this is funny! And thanks to Mark Mitchell for repeating this advice frequently <http://en.wikipedia.org/wiki/Marathon>.

Can You BACKTALK?

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

For a complete author's packet detailing how to submit your BACKTALK article, visit our Web site at <www.stsc.hill.af.mil>.

NAVAIR'S Strategic Priorities

Current Readiness

Contribute to delivering Naval aviation units ready for tasking with the right capability, at the right time, and the right cost.

Future Capability

Deliver new aircraft, weapons, and systems on time and within budget, that meet Fleet needs and provide a technological edge over our adversaries.

People

Develop our people and provide them with the tools, infrastructure, and processes they need to do their work.

NAVAIR Software/Systems
Support Center
Comm 760 939-6226 DSN 437-6226

CrossTalk is co-sponsored by the following organizations:

NAVAIR

Homeland Security

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

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