MANAGEMENT CHALLENGES AND TECHNIQUES ON A LARGE ADA PROJECT

Capt. Richard W. Root, USAF/ESD
Mr. Gerard LaCroix, Mitre Corporation
Mr. Michael Springman, TRW Systems Integration Group

Abstract

The Command Center Processing and Display System Replacement (CCPDS-R) is a large U.S. Air Force Ada application being successfully developed using an incremental development and test process. The first subsystem delivery was made to the Air Force in December 1990, consisting of over 280,000 Ada source lines of code, operator display consoles, and data processing equipment. Since contract award in June 1987, a number of management challenges have been addressed by the Government procurement agency (Electronic Systems Division, with support from the MITRE Corporation and SDAS contractor) and the developing contractor (TRW Systems Integration Group). This paper discusses management actions taken to address key issues, such as:

1. How to obtain visibility into true software development progress.
2. How to run an efficient formal test program as new increments of capability are produced and integrated.
3. How to achieve and maintain software productivity that is higher than industry averages for real time systems.
4. How to buildin quality throughout the incremental development process.
5. How to retain and motivate the software staff on a multiyear development project.

Various aspects of the incremental development approach, features of the Ada language and support environment, and characteristics of the message-based Ada design are discussed as they relate to the above issues. Topics include the use of design walkthroughs; formal demonstrations of functionality; top-down integration; early development of the architectural and system service software base; delivery of operational capability via series of tested software builds; software management metrics; trades between CPU utilization and throughput; creation of software tools to aid the development process; standardized task-to-task communications; reuse of design, code, and people; flow-down of award fees; timely commitment to requirements and design assumptions and the content of incremental formal tests; management of concurrent developments; and minimization of breakage as new functional increments are incorporated.

Background

System Description. CCPDS-R provides missile warning information within the Cheyenne Mountain Complex (CMC) for the North American Aerospace Defense (NORAD) Command. CCPDS-R processes and displays Integrated Tactical Warning/Attack Assessment (TW/AA) data for the National Command Authorities and allies. Satellite sensors and strategically located ground-based sensors transmit messages to the Integrated TW/AA System notifying the National Command Authorities of any attack on the United States or its allies. CCPDS-R serves as the central warning system for any attacks and gives the National Command Authorities the information necessary to make the appropriate retaliatory decisions. CCPDS-R processes the sensor messages and displays the results on geographic and tabular formats.

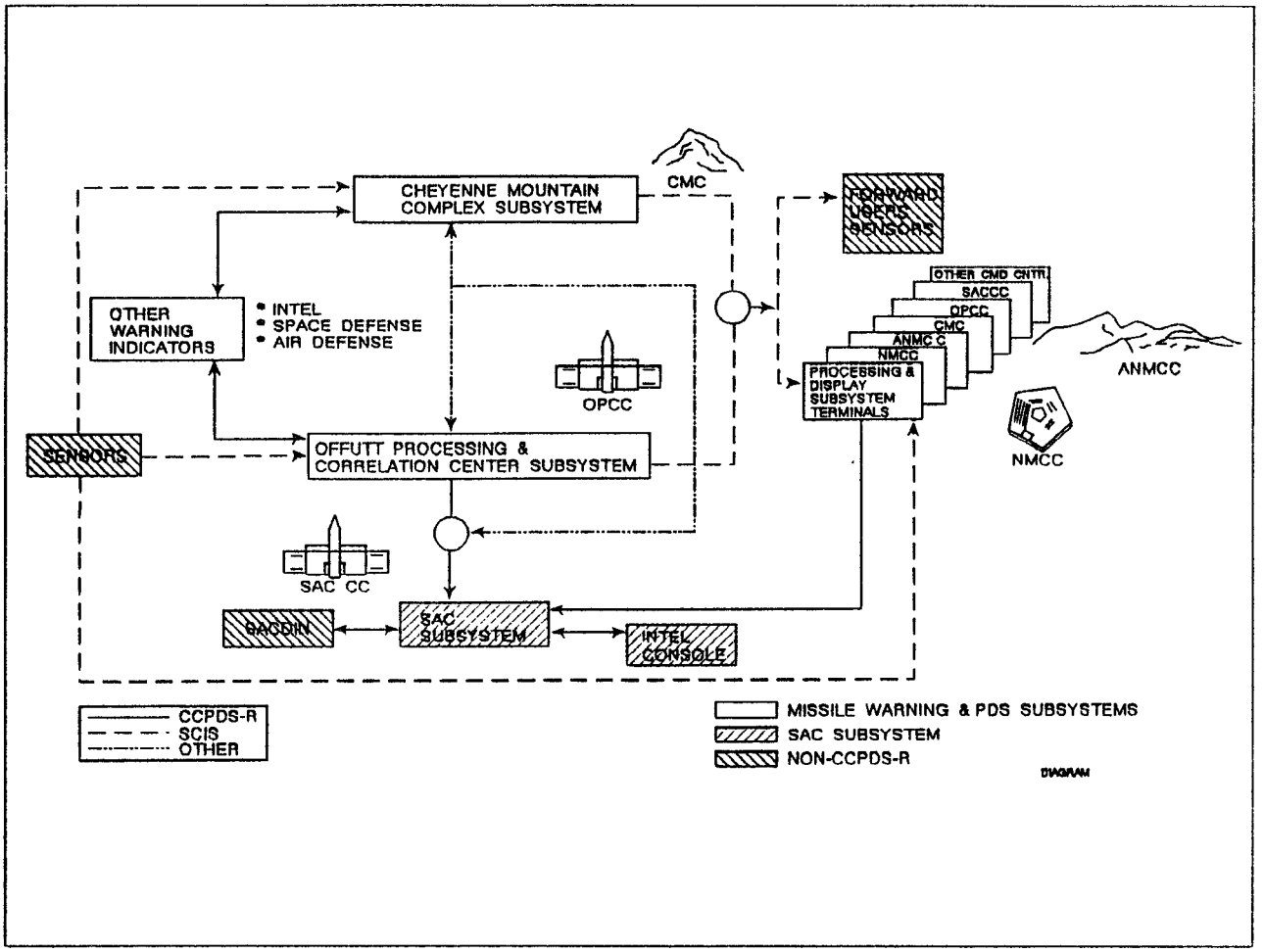


Figure 1: CCPDS-R System Diagram

futt Air Force Base, with an identical subsystem installed in the SAC Development Test Facility for software maintenance and support. The SAC Subsystem, in addition to the Missile Warning Subsystem capabilities, has additional functions for Force Management/Force Survivability. The third subsystem is the Processing and Display Subsystem (PDS), which will be placed at various locations throughout the world in support of the Joint Chiefs of Staff, Nuclear Capable Commanders-in-Chief, and Canadian Command Centers. The PDS performs a subset of the Missile Warning Subsystem's processing. Its main function is to display to worldwide commanders the same information that is being displayed at the primary correlation centers (CMC or OPCC).

Development Approach. There have been six increments (builds) of software for the Missile Warning Subsystem. A key objective during the MW development has been to maximize the amount of software that is reusable for the SAC subsystem and the PDS. Each increment of software employed a prototype and build approach which, in essence, breaks the overall subsystem development cycle into a series of smaller, more-easily manageable, development increments [Royce 1990-1]. This approach is characterized by three milestones: Preliminary Design Walkthrough (PDW), Critical Design Walkthrough (CDW), and formal demonstration of the operational build contents.

Using the prototype and build approach (Figure 2), CCPDS-R software development began significantly earlier than would have been the case with a more traditional software development. By the time of the Missile Warning Subsystem Critical Design Review (CDR), over 75% of the code had been designed, integrated, and informally tested. The earliest software developed consisted of the architectural and system services software base upon which all subsequent applications software would depend. Thus, integration, which is historically the riskiest part of software development, has been an

early and continuing effort on the CCPDS-R Program. Traditionally, integration does not begin until well after CDR. As a result, integration problems tend to surface late in the development when their effects are magnified by the pressures of schedule and the contention for resources.

Progress Measurement. The Government and the Contractor have instituted several useful techniques for determining software development progress on CCPDS-R [LaCroix 1991], including:

1. Evolving demonstrated prototype code into the operational configuration.
2. Meaningful design walkthroughs.
3. Early development of architectural foundation software.
4. Incremental delivery of operational capabilities via software "builds".
5. Formal demonstrations of functional capabilities.
6. Monthly comprehensive assessment of progress via software management metrics.

To date, for example, the Missile Warning Subsystem has had 6 PDWs, 6 CDWs, and 5 formal demonstration-oriented requirements/design reviews, in addition to various informal demonstrations. The Government and TRW management receive quantitative summaries of progress via the regularly collected and reported software management metrics. Actual versus planned progress is tracked for software staffing, software size, development progress (Figure 3), integration progress, and test progress. Separate accounting is maintained for the percentages of the software which have been designed, coded, standalone tested, integrated, string tested and documented. Similar metrics are tracked for test procedures completed and requirements verified (Figure 4). Cumulative plots of open and closed software problem reports and software documentation problem reports are generated monthly. The status of all action items resulting from formal meetings is also updated monthly, along with plots of cost and schedule variances. Recent examples of progress metrics for the PDS Subsystem are included in Figure 4.

Government Role in Development Process

Visibility is a key to determining the real status of software development efforts. Visibility becomes even more important when the software being developed is large, complex, and time-critical, as is the case for the CCPDS-R Program. The Government's experience to date is that there has been much more visibility into true software development progress for CCPDS-R than other current and past programs. This visibility is attributable to several factors.

Requirements Feedback. As discussed earlier, the CCPDS-R Program employs a prototype and build approach. By developing a prototype early, code is designed, integrated, and informally tested prior to any formal reviews. This gives the Government an early opportunity to actually see how the contractor's design proposes to meet specific requirements, and provides a vehicle for immediate feedback to the contractor. This enables requirement interpretation issues to be surfaced early, minimizing downstream requirement changes.

Design Walkthroughs. The PDW is an informal technical design walkthrough that the contractor conducts to review actual design products (prototypes, Ada Design Language (ADL), graphic depictions). The CCPDS-R walkthroughs are highly interactive, with the Government (including the Users) and other program organizations (e.g., System Engineering, Hardware Engineering, Integrated Logistics) attending and actively participating. The walkthroughs are a more effective vehicle than formal design reviews for attendees to really understand the design concepts and determine if the design products will satisfy the requirements. Demonstrations of prototyped components complement the design presentations and enable the audience to actually see the design in action. All questions and issues raised during the walkthroughs are documented and formally tracked to ensure timely resolution. Once it is agreed that the top-level design is complete and will satisfy the requirements allocated to the build components under review, the contractor continues design and development to meet the next milestone, CDW. At the CDW, again the Government and the contractor review the detailed design products for the build components, and observe their proposed operation via demonstrations.

After the PDW is conducted for the final subsystem build, the formal subsystem-level PDR is conducted. Because the software top-level design has been exhaustively reviewed at the PDWs, the PDR presentation concentrates on requirements issues and system-level design issues, with a summary of what transpired at the PDWs. The highlight of the PDR is a comprehensive formal demonstration of the components prototyped/developed and integrated to date, which clearly shows the PDR attendees how the system is being implemented. Similarly, the subsystem-level CDR is

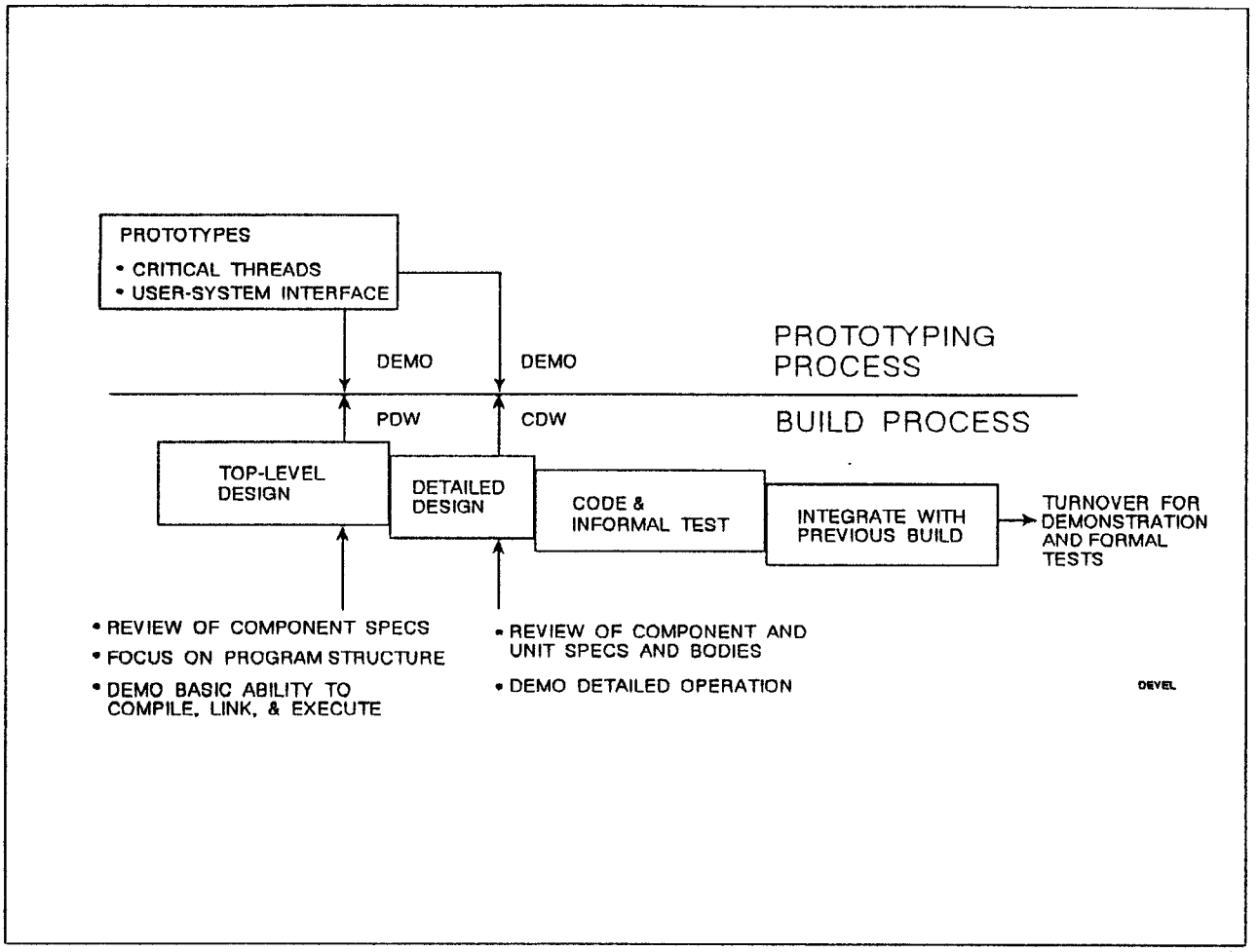


Figure 2: Prototyping and Build Processes

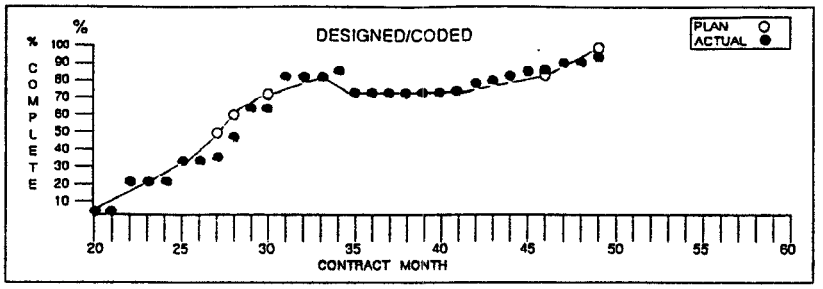
conducted following the final build CDW, with a formal demonstration again highlighting the activities. The formal demonstrations enable the Government to "touch and see" the results of the design. By the time the CDWs and the formal demonstrations are complete, most of the code (over 90% for the MW Subsystem) has been developed and integrated, and is in a working state (although not yet completely formally tested).

Early Architecture Validation The earliest builds of software developed by the contractor consisted of the architectural and system service software base. This base software is the integration framework for all the various tasking done in the CCPDS-R Program. This base, consisting of the Network Architecture Services [Royce 1989] and the software architecture skeleton, included reusable node managers, process executives, and task executives. These components permit applications developers to concentrate on the functionality associated with their respective modules without having to know the inner workings of the software infrastructure and the mechanisms by which Ada tasks communicate with other Ada tasks. Thus, the internal integration of all the different tasks that needed to be accomplished for CCPDS-R was completed early in the program.

This is a significant problem that most Government programs currently have in the software acquisition process. Typically, a large software-based system consists of thousands of requirements that the Government specifies and the contractor refines. The contractor begins to concentrate on the individual requirements and fails to realize the big picture: the integration of all the individual requirements into the total system. This is very evident when you look at Air Force software programs that eventually purchase new hardware and completely redesign the original software to incorporate minor additional tasks or requirements. By the contractor establishing the architecture early, the Government can

CSCI	IMPLEMENTED TOTAL KSLOC	Complete (Ada)	%	TBD (ADL)	CM ADL→ADA	TURNED OVER TO SWENG PLANNED	TURNED OVER	%	CM △ T/O
NAS	18.3	18.3	100%	0	0	18.3	18.3	100%	0
TAS	9.9	9.9	100%	0	0	9.8	9.9	100%	0
PSSV	138.6	130.6	94%	8.0	1.5	120.1	130.7	94%	2.8
PDCO	44.2	38.7	88%	5.5	.5	43.1	37.5	85%	5
PCO	26.6	26.4	92%	2.2	4.2	17.0	21.9	77%	.5
TOTAL	239.6	223.9	93%	15.7	6.2	208.3	218.3	91%	3.8

* TURNED OVER ÷ TOTAL KSLOC



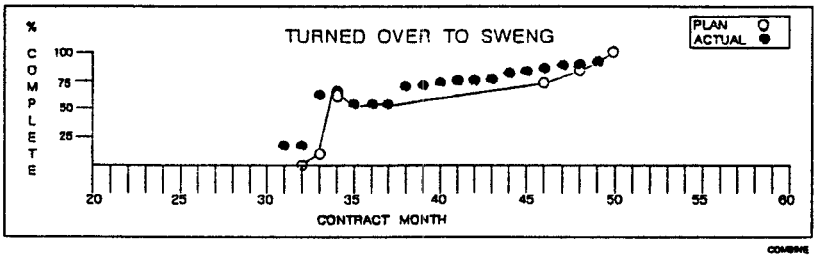


Figure 3: Example Development Progress Metrics for PDS

validate and approve the architecture, including growth and flexibility considerations, and enable concentration on implementing and verifying the individual requirements. The integration, which is usually the most complex portion of a software program, is accomplished earlier and incrementally, lowering the risk of downstream integration problems (see Figure 5).

This solid architecture foundation has provided the Government with software that exhibits high quality and reliability (i.e., meantime between critical failures). Even though the contractor is still completing development, reliability testing conducted by the contractor predicts a software reliability in excess of 10,000 hours.

This reliability is attributable not only to the early robust architecture builds, but also to the tools that the contractor has developed to generate code which would otherwise be labor-intensive and repetitive, and hence prone to human error. For example, the CCPDS-R Program uses a variety of tools to generate compilable Ada code from ASCII and pseudo-English data inputs. The principal tools (Figure 6) consist of 30,000 Source Lines of Code, and are used to generate over 300,000 operational source lines of code or database records. This is an expansion factor of 10 to 1. Verifying the proper operation of the tools that generate the code products is easier, less time consuming and less error prone than manual verification of all of the tool generated code itself.

Software Test Approach. The CCPDS-R formal software test program was defined early, involving extensive, coordinated effort by the Government and the contractor to devise a comprehensive, efficient test concept that complemented the incremental development process [Springman 1989]. The resulting test program is multi-level, including Standalone Tests (SATs) for CSC-level requirements verification, Engineering String Tests (ESTs) for integrated CSCI requirements verification,

TEST PHASE	Requirements Verification Planned/*Completed*						
	NAS	PSSV	PDCO	TAS	PCO	QPRs	TOTAL
Previously Verified (Reused CSCIs)	*237*	0	0	*221*	0	0	*458*
Build P1 SAT	0	69	66	0	25	0	160
EST1	0	21	101	0	6	0	128
EST2/FQT	0	158	314	3	53	12	540
TOTAL	237	248	481	224	84	12	1286

Figure 4: PDS Software Requirements Verification Metrics

and Formal Qualification Test (FQT) for verification of requirements usnng the full system software and hardware configuration. For the MW Subsystem, there was a formal SAT phase and a formal EST phase associated with each major incremental software build, with a single FQT at the end.

The Government's role in the test program is critical. The Government must witness all formal requirement verification activities, accomplished per approved test plans and procedures. Government test reviewers must interact closely with contractor test engineers to ensure that procedure problems and requirement issues are resolved in as timely a manner as possible. Otherwise, the ongoing incremental development and test schedules may be severely impacted, creating an undesirable "bow wave" of deferred test cases. Because the testing is incremental, with about 80% of the requirements verified during SAT and EST phases, the number of requirements to be formally verified at FQT is kept to a level that is manageable by both Government and the contractor.

Management Metrics. In addition to the formal reviews and demonstrations, the productivity or progress of the program has been monitored by monthly quantitative software metrics reports [Andres 1990]. As described earlier, these reports track a number of issues/resolutions over the entire program. The Government has used the metrics report to provide insight into difficulties the contractor is encountering. A specific example occurred when the communications portion of the software was exceeding performance budgets after completion of development and turnover to the integration group. The metrics showed that additional manpower was being allocated to this portion of code and that the performance parameters were showing excessive resource utilization. It signalled the Government to closely monitor the contractor's actions to resolve the problem, and to suggest alternative actions themselves.

As new metrics or ways to report metrics are discovered, both the Government and the contractor have tailored the standard metrics reporting to incorporate improvements. This ensures that the metrics set remains as useful for management purposes as possible.

Contractor Role in Development Process

Manageable Software Increments. The contractor has the difficult task of developing a large, complex program, verifying it against the Government's specifications, successfully passing all reviews and audits, and delivering the entire program to the Government's satisfaction. By specifying incremental builds of software, the application developers and testers focus on smaller, more manageable sets of requirements. After initial development was completed, the three milestones (PDW, CDW, formal demonstration) became the primary means of solidifying the Government requirements and the developed code. The primary purpose of each of the three milestones was to enable the Government to "touch and see" the working software, review the design, and assess its satisfaction of the Government's requirements in a working-level environment. This allowed the Government to confirm/modify the requirements that were being addressed, clear up any unresolved issues or disconnects, and provide a focus for both the Government and the contractor for future builds and developments. This process was the major means to communicate design progress and results to the Government, ensuring that the development was addressing the requirements and needs of the program.

Robust Architecture Foundation Components. The early development of the architectural/system service software ensured its early availability for use by the applications programmers, and its continuous, repetitious use contributed to its inherent reliability. The software team was formed around a highly knowledgeable team of Ada experts and software architects, augmented by experienced applications developers organized into manageable skill groups. The

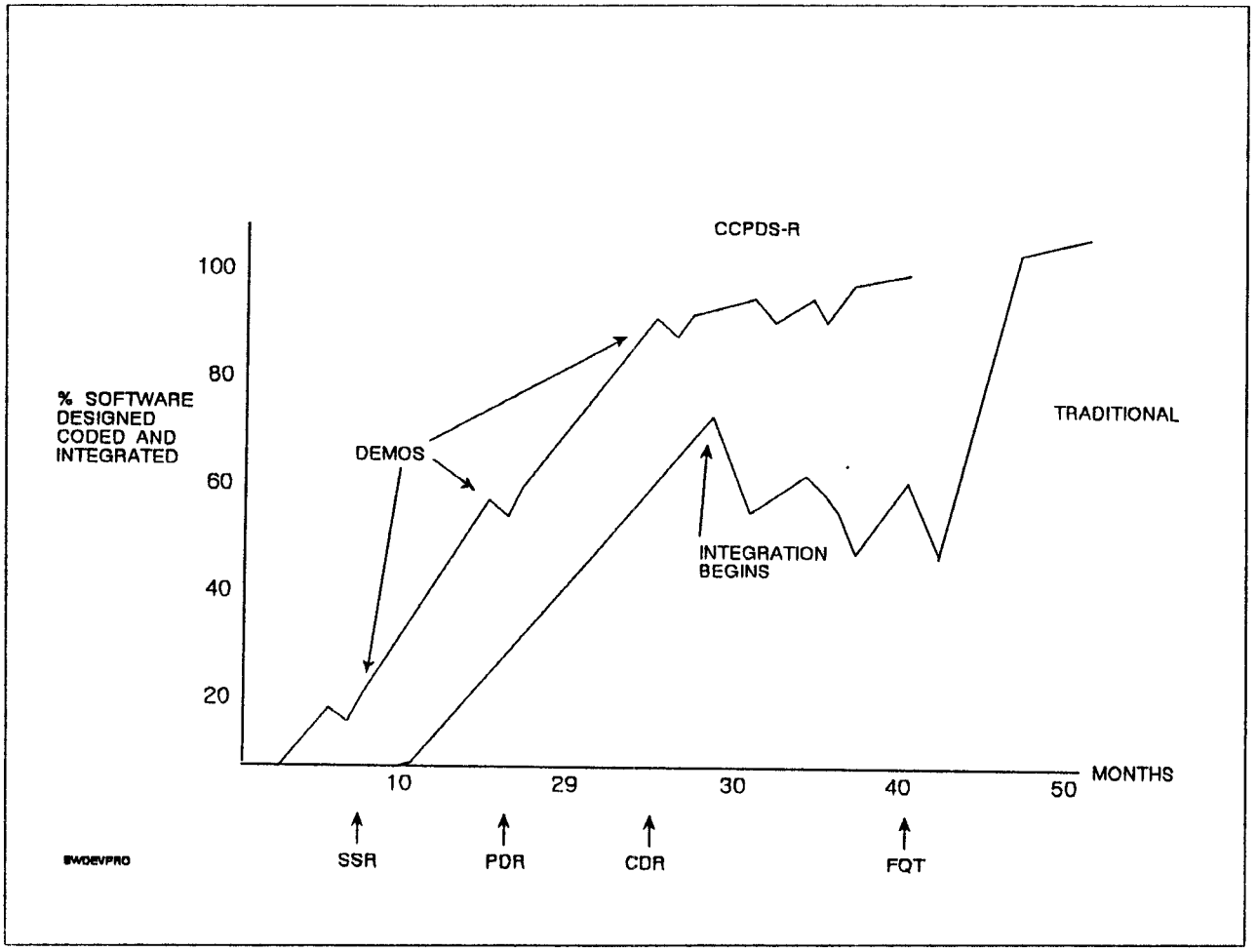


Figure 5: CCPDS-R Software Development and Integration Approach

team of Ada experts and software architects has been responsible for the more difficult software code, i.e., the architectural/system service software. This code effectively isolates the applications developers from the complex Ada constructs of tasking, rendezvous, time-slicing, prioritization, etc. The net result is code appropriate to the technical abilities of the individuals and hence reduced errors.

By establishing the architecture early, standard development procedures were implemented. These include standard ways for developers to interface their applications modules with the reusable architectural software components, standard ways for writing applications code using templates at both the application "process" and "task" levels, and standard style guidelines. This results not only in fewer errors (3.5 per 1,000 Source Lines of Code versus 5-12 per 1,000 Source Lines of Code industry-wide) but also in high maintainability [Royce 1990]. The result is less labor to fix problems and hence higher productivity. On the CCPDS-R Program, about 2/3 of the software errors have required less than one day to fix. (It should be noted that the CCPDS-R software is still being maintained in the contractor's development environment. The Government has not yet assumed maintenance responsibility.)

Using Metrics as Problem Indicators. The software metrics provide clear indications concerning specific problems that need to be addressed. For example, as the development of the CCPDS-R software progressed, the metrics showed that the performance margins were being exceeded, signalling to management that action was required. The choices were twofold: (1) optimize the software to make it more efficient or (2) upgrade the processors. Optimizing the software is labor-intensive and reduces productivity. Nevertheless, on a few occasions software optimization efforts were undertaken to speed response times and increase throughput. On other occasions, the decision was made to upgrade

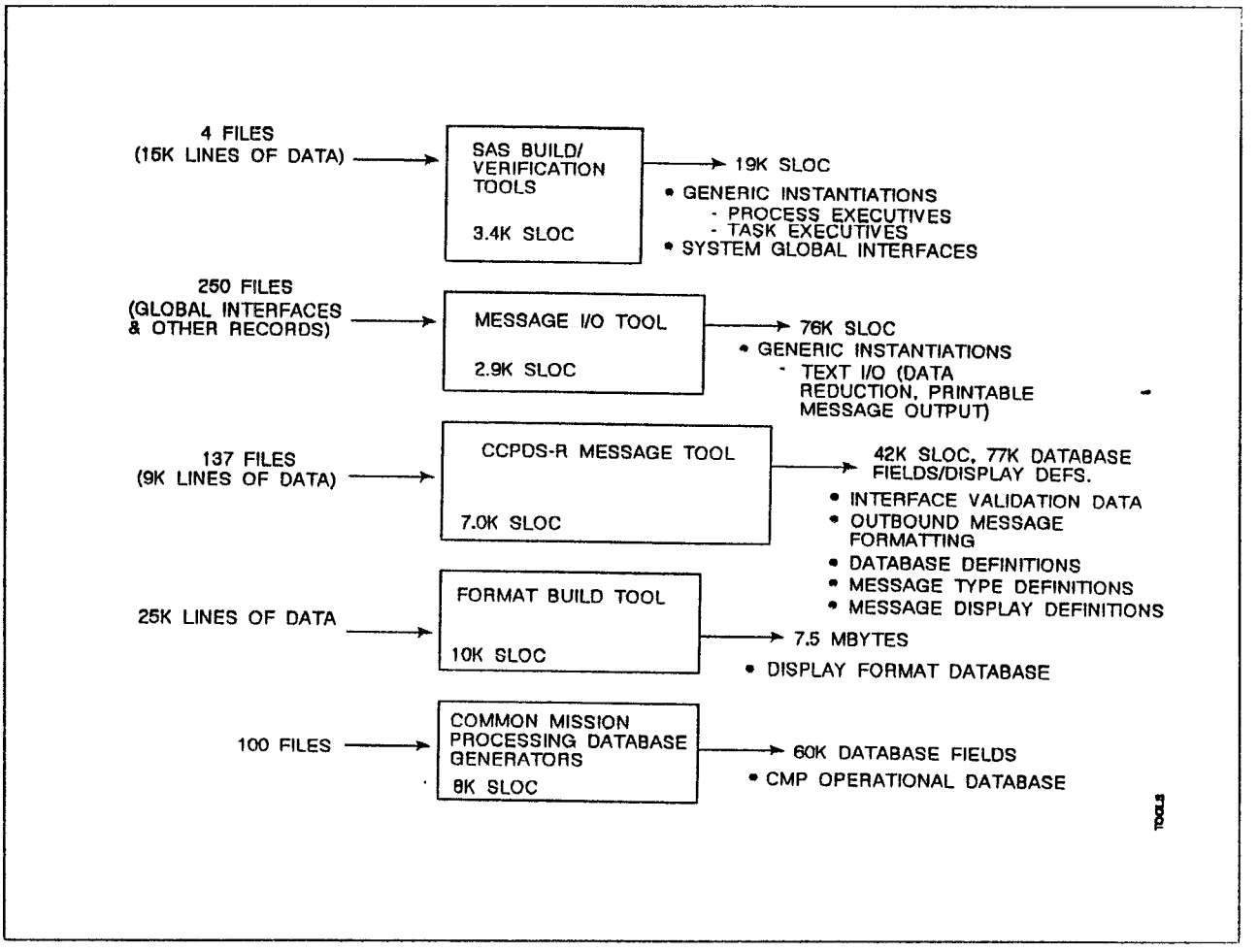


Figure 6: Tool-Generated Code/Databases

the processors because of two factors. First, the software could be ported to the upgraded processors with virtually no changes and hence no labor expenditure. Second, the upgraded processors could be purchased at about the same cost as the originally planned processors due to continuing advances in processor technology. In either case, the metrics flagged management early enough so that the performance problems could be addressed with minimal financial or schedule impact to the program.

Another problem identified from the metrics report was the growth in requirements, resulting in decreased benefit of the incremental testing methodology. This became apparent when the number of requirements that would have to be formally tested at FQT (which is a short two week period) kept growing to an unmanageable level. To correct this, management decided to move requirements to newly created formal Stand Alone Test and Engineering String Test phases to reduce the number of requirements to be verified at FQT.

Using Metrics for Productivity Tracking. The metrics reports readily show where progress is or is not being made, and enables management to continuously track team productivity. This program is experiencing software productivity in the range of 7 to 14 Ada Source Lines of Code per staff-day, depending on the subsystem. Productivity ranges are a function of how the code is counted, which is complicated by differences in the amount of effort required to produce new code, modify previously developed code, reuse existing code, and generate new code using automated tools. The computation is further obfuscated by the lack of industry standards for counting code (the numerator of the productivity equation) or for what labor to include (the denominator).

The approach taken on CCPDS-R to compute productivity is:

1. Weight new code and modified code equally (only lines of code actually generated or modified are counted).

2. Weight reused and tool generated code based upon the relative effort required to generate it when compared to the effort required to generate new code. This results in a multiplicative factor ranging generally between .3 and .8 for reused and tool-generated code.

3. Count Source Lines of Code by totalling carriage returns in Ada Specifications and semi-colons in Ada bodies; do not count "Comment" lines. Design and coding standards dictate that enumerated type, record type, and subprogram declarations be declared with a single type/field/parameter per line. This provides a more realistic accounting of the engineering effort associated with defining complex and/or voluminous data types.

Applying this approach to the Missile Warning Subsystem code yields a weighted software productivity of approximately 7 Ada SLOC per staff-day, which is about 40% higher than the usually quoted industry figure of 5 Ada SLOC per staff-day. The major factors contributing to this relatively higher productivity are reuse of code and use of tools to generate code. In fact, on the CCPDS-R Program, 35% of the developed and tool-generated code is being reused from one subsystem to the next, and 40% of the over one million total developed SLOC is being generated by means of tools.

The productivity rates for the PDS and SAC subsystems are projected to be substantially higher than for the MW subsystem because of the reusability of the software among subsystems, the maturity of the code generation tools, and the experience of the Government/contractor team with Ada and the tools. The weighted productivity for PDS is projected at 14 Ada SLOC per staff-day, with SAC projected at 10 per staff-day.

Recent experience with the SAC software at the SAC Preliminary Design Review shows the value of the reusable software, flexible architecture, and incremental development approach. Over 75% of the SAC Subsystem software, which is estimated to be a total of 420,000 Ada SLOC, was integrated in less than 2 months for the highly successful formal SAC Subsystem Demonstration conducted at PDR.

Staff Retention. CCPDS-R was one of the first major Ada projects undertaken by the Government team and the contractor. Management on both sides was acutely aware that true software successes on large programs were few and far between, with a major reason being excessive personnel turnover as the program progressed. To incentivize people to stay with the program, especially the top performers, the contractor is flowing down half of any award fee received from the Government directly to the contractor employees working the program. The award fee potential, the attraction of the application, the opportunity to work on a well-managed Ada program, and a desire to be associated with a success have all been factors in the contractor's ability to motivate and retain staff.

Summary

When comparing the CCPDS-R Program to other programs which have had successful software developments (such as the Berlin Radar Program), certain similarities stand out. These include the quality of the documentation and test program, the close professional working relationship between the contractor and Government teams, knowledge by both of the job to be done and how to go about doing it, and continuity of key personnel. Such are the characteristics of a good, well-run program.

Two additional factors contribute to make the CCPDS-R Program an exceptional program:

1. The use of a flexible architecture characterized by reusable architectural components, standard task-to-task communications, and standard applications process and task templates.

2. The use of an incremental Ada software development process model consisting of operational code prototypes, frequent design walkthroughs, incremental builds and testing, and demonstration-oriented design reviews.

The use of a message-based design relieves the applications developers from the burden of creating software for task-to-task communications; early integration of the architectural/system service software and applications shells allow for their reliable use and reuse throughout the development cycle; and continuity of knowledgeable staff, primarily within the contractor organization but also on the Government acquisition team, ensures incorporation of lessons learned for subsequent development activities.

In conclusion, the CCPDS-R software development approach is working. Performance, schedule, and cost objectives are being met. The success enjoyed to date

has been due to a combination of the capabilities provided by Ada technology, the software development approach, and the dedication and expertise of the Government and contractor personnel. The constant communication between the Government personnel and the contractor personnel can not be emphasized enough. By having this constant interchange of information, both parties come away with a clear understanding of the job.

This software development approach has resulted in an environment characterized by a high degree of visibility into true software development progress, high software reliability, and high software productivity. The CCPDS-R Program is a model for others to follow.

Biographies

CAPTAIN RICHARD W. ROOT is the Software Chief Engineer for the CCPDS-R Program, U. S. Air Force, Electronic Systems Division, Hanscom Air Force Base, Massachusetts. Capt. Root is responsible for the entire system software (over 1,000,000 Ada Source Lines of Code) including the design's technical approach, architecture integrity, system performance, and reusability. Prior to this position he was a lead engineer working on the B-1B aircraft responsible for aviation electronics, flight software, survivability and vulnerability assessment, and all future modifications to the aircraft. He received a BS in Electrical Engineering (Computer Engineering) from the University of Wyoming in 1985 and is currently working on an MS in the same field.

GERARD LACROIX is the Project Leader of the CCPDS-R Program, MITRE Corporation, Bedford, Massachusetts. He is responsible for the overall MITRE system engineering support to the program. Prior to June 1987, he was a member of the MITRE Software Center and one of its founders. In 28 years at MITRE, he has worked on a wide variety of strategic and tactical Command, Control, and Communications (C^3) systems, including BUIC, AWACS, NADGE, Seek Dawn, Igloo White, MCE, GSTDN, and TDRSS. He received an MS degree in Electrical Engineering from Carnegie Institute of Technology in 1963 and a BS degree from Lowell Technological Institute in 1962.

MICHAEL SPRINGMAN is TRW's Deputy Program Manager for CCPDS-R, responsible for all software activities. He has been involved with the program since its inception. His prior experience during 15 years at TRW includes all aspects of the system/software development cycle for a variety of C^3 and avionics systems. He received an MS in Applied Mathematics/Computer Science from the University of Colorado in 1975 and a BA in Mathematics/Physics from Southwest (MN) State University in 1973.

References

[Andres 1990] Andres, D.H., "Software Project Management Using Effective Process Metrics: The CCPDS-R Experience", *AFCEA Military/Government Computing Conference on Software Engineering*, Washington, D.C., January 1990.

[LaCroix 1991] LaCroix, G.R., "Experience with an Incremental Ada Development in Terms of Progress Measurement, Built-In Quality, and Productivity", *Analytical Methods in Software Engineering Economics Conference, MITRE Economic Analysis Center*, McClean, VA, 29-30 April 1991.

[Royce 1989] Royce, W. E., "Reliable, Reusable Ada Components for Constructing Large, Distributed Multi-Task Networks: Network Architecture Services (NAS)", *TRI-Ada '89 Proceedings*, Pittsburgh, October 1989.

[Royce 1990-1] Royce, W. E., "TRW's Ada Process Model For Incremental Development of Large Software Systems", *Proceedings of 12th International Conference on Software Engineering*, Nice, France, 26-30 March 1990.

[Royce 1990-2] Royce, W. E., "Pragmatic Software Quality Metrics for Evolutionary Software Development Models", *Tri-Ada '90 Proceedings*, December 1990.

[Springman 1989] Springman, M. C., "Incremental Software Test Approach For DOD-STD-2167A Ada Projects", *TRI-Ada '89 Proceedings*, Pittsburgh, October 1989.