# Ada AND THE RAPID DEVELOPMENT LIFECYCLE

Lloyd DeForrest
Dr. Lynn Gref
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91009

## ABSTRACT

JPL is under contract, through NASA, with the U.S. Army to develop a state-of-the-art Command Center System for the United States European Command (USEUCOM). The Command Center System will receive, process and integrate force status information from various sources and provide this integrated information to staff officers and decision makers in a format designed to enhance user comprehension and utility. The system is based on distributed workstation class microcomputers, VAX- and SUN-based data servers, and interfaces to existing military mainframe systems and communication networks.

JPL is developing the Command Center System utilizing an incremental delivery methodology called the Rapid Development Methodology with adherence to government and industry standards including the UNIX operating system, X Windows, OSF/Motif and the Ada programming language. Through a combination of software engineering techniques specific to the Ada programming language and the Rapid Development Approach, JPL has been able to deliver capability to the military user incrementally, with comparable quality and improved economies of projects developed under more traditional software intensive system implementation methodologies.

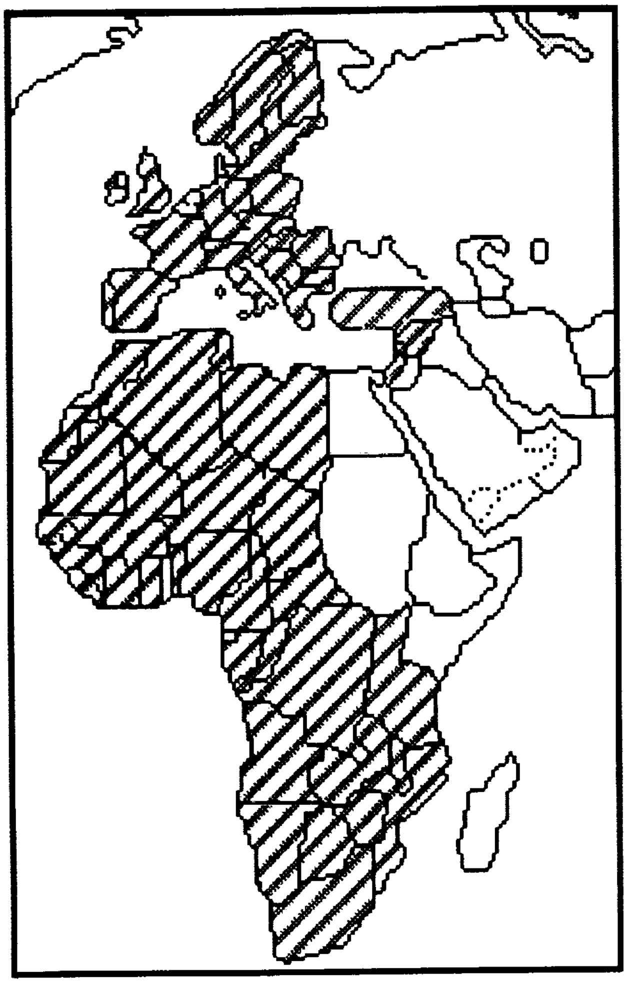
## INTRODUCTION

The objective of the EUCOM Command Center Project (ECCP) is to provide USEUCOM a state-of-the-art Command Center System which integrates information available from approximately one hundred communications and automation systems. JPL is designing, implementing, installing, and testing this "core" system. The system will serve the Command staff in their mission execution. Figure 1 illustrates USEUCOM's area of responsibility and its basic mission. The area of responsibility covers all of western Europe and includes most of Africa and the Mediterranean Sea to the Suez Canal. The Command has responsibility for all unilateral actions of the U.S. military within its area of responsibility. Many world events of the past two years have involved USEUCOM. Disintegration of the Soviet Block, the Gulf War, the hostage releases in Lebanon and the U.S. Kurdish relief effort are just a few of these events affecting the Command. As a consequence, with each new crisis action, new requirements are levied on the Command Center System to process and present force status information in new or different ways. The impact of these new requirements is compounded by a shrinking budget for implementation of the Command Center System that has resulted in the demand to do more for less.

The following general functional capabilities have been identified as the minimum set that the Command Center System requires:

1. Available information shall be passed, processed and/or aggregated in a timely manner.
2. Information shall be automatically distributed, plotted and correlated.
3. Status and location of all forces shall be consolidated, integrated, and presented.
4. Presentation of available information shall facilitate the decision making process.
5. Integrated access to data from multiple security levels will be made available at the earliest opportunity.
6. Communications facilities shall support secure automatic voice, data and teleconferencing.

The overall architecture for the core system which will provide the above cited functional requirements is depicted in Figure 2. This architecture recognizes the need for an integrated approach for the three mediums of information: visual (video), voice, and digital data. A communications network for each security level and each medium



- Unified Command
- Area of Responsibility (AOR)
  - Europe, Africa, Mediterranean
  - 13 Million Square Miles
  - 350,000 U. S. Troops
- Strategic and Operational Tasks
  - Conventional, Nuclear, Special
- Continuus Operations
  - Peacetime
  - Crisis Management
  - Warfare
- Post-Conventional Forces Europe Functions
  - Monitoring / Directing Out-Of-Area Operations
  - Monitoring Compliance With Force Reduction Treaties
  - Coordinating Expanded Intel And Recon Services
  - Arranging / Directing Rapid Force Reduction

Figure 1. U.S. European Command Area of Responsibility and Mission

interconnects the various offices (i.e. sections) of the Command. Data Local Area Networks (LAN) of different security classification levels are interconnected by specialized security gateways. These gateways initially provide data transfers in one direction, from a lower security classification to a higher security classification. Future generations of the security gateways will provide limited two way data transfers.

The U.S. Army has adopted a number of standards for the implementation of Command Center Systems. These standards are designed to promote the sharing and reuse of software among Command Center Systems. Figure 3 depicts the layered software architecture with the associated standards being followed by the Project. In addition, the Project's documentation follow that required by the DoD-STD-2167A [7]. The adherence to these standards have presented many technical challenges with regards to software development. For example, bindings between the Ada programming language and the X Window/MOTIF user interface had to be developed. This software required a moderate effort to develop (approximately 15,000 lines of code and two work years) and has been provided to various military and commercial organizations with military sponsor approval (we are now in the process of placing this software into COSMIC). Another example is the development of a set of Application Support Layer libraries which facilitate the current and future development of user interface, map access, decision support and monitor and control applications.

Further challenges to the software development within the ECCP are: 1) a constrained and varying funding profile, 2) adherence to newly adopted standards, 3) utilization of Commercial Off-The-Shelf (COTS) and Non-Developmental (NDI) software whenever possible, 4) development of software in Ada, 5) employment of incremental deliveries every 9 to 12 months, and 6) the rapid turnover of user community members (users involved with the requirements definition and design are frequently not the actual users of the operational system).

The remainder of this paper provides a description of our approach to solving these challenges through a discussion of the Rapid Development Methodology, a brief discussion of the Ada programming language (given as background information), a discussion of the ECCP's combination of Ada and the Rapid Development Methodology, a discussion of the benefits of this approach (with some pitfalls), and a discussion of some of the lessons we have learned that might be beneficial to others.

## RAPID DEVELOPMENT METHODOLOGY

The challenges briefly described in the previous paragraphs necessitate an unconventional development methodology. It has been our experience that the conventional wisdom upon which the classic "waterfall" approach [1] depends requires the following conditions be met, or at least maximized, for a successful project development:

1. Requirements be identified and configuration managed early in the development cycle [ref. 1, pg. 39]. This requires that the user articulate program behavior requirements sufficiently before the design phases.
2. A budget and funding profile be established up-front and followed, minimizing perturbations in Project team staffing.
3. Users can forgo getting the new capabilities for the 3 to 8 year development cycle.
4. The systems on the other side of external interfaces will not change unexpectedly during the development.

The Incremental Waterfall [1], and the Spiral Model [2,3,4] methodologies are two examples of methodologies that address, and attempt to minimize, the effects of change to the software development induced by the above conditions not being met. Additionally, JPL has developed a software engineering methodology similar to the Waterfall and Incremental Waterfall methodologies called the "JPL Software Management Standards" [5].

The Rapid Development Methodology (RDM) is similar to the Incremental Waterfall, Spiral and institutional JPL methodologies, but has been tailored to specifically address the above mentioned conditions. RDM has been used successfully on previous JPL command center improvement projects (e.g. the Distributed Management Information and Communication System {DMICS} and the Global Decision Support System {GDSS}).

Another variation of the incremental development methodologies is the incremental delivery of prototypes (rapid prototyping) which could be used (as was used during the early stages of the DMICS) to continually establish and

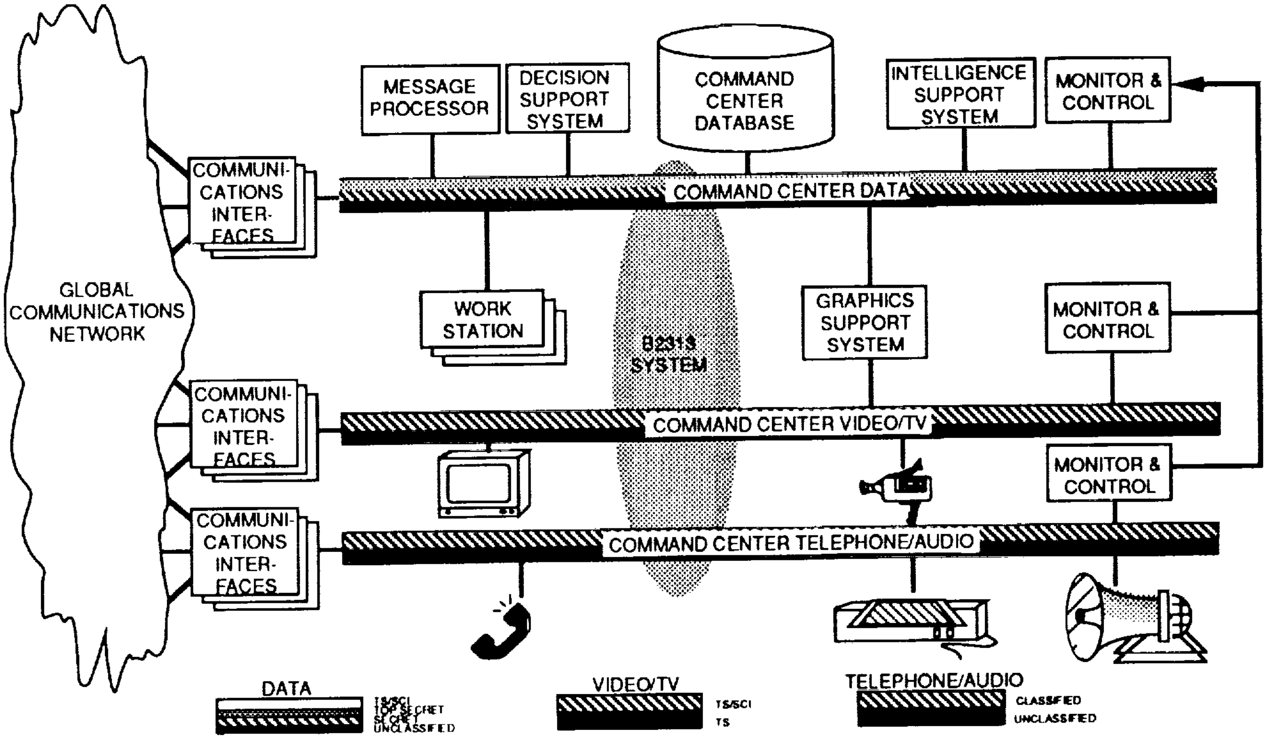
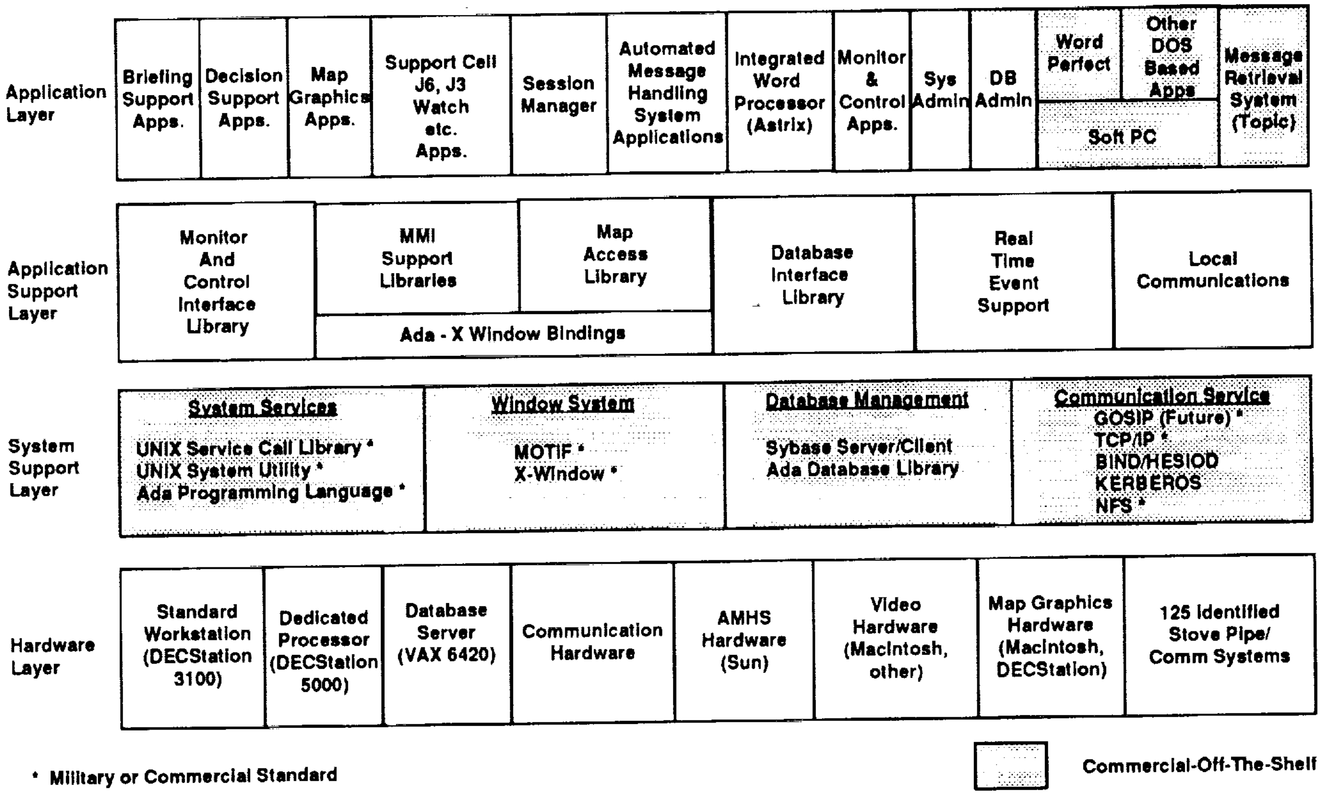


Figure 2. Generic Command Center Architecture



| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Application Layer | Briefing Support Apps. | Decision Support Apps. | Map Graphics Apps. | Support Cell J6, J3 Watch etc. Apps. | Session Manager | Automated Message Handling System Applications | Integrated Word Processor (Astrix) | Monitor & Control Apps. | Sys Admin | DB Admin | Word Perfect<br>Soft PC | Other DOS Based Apps<br>Soft PC | Message Retrieval System (Topic) |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Application Support Layer | Monitor And Control Interface Library | MMI Support Libraries<br>Ada - X Window Bindings | Map Access Library<br>Ada - X Window Bindings | Database Interface Library | Real Time Event Support | Local Communications |

| | | | | |
|---|---|---|---|---|
| System Support Layer | System Services<br>UNIX Service Call Library *<br>UNIX System Utility *<br>Ada Programming Language * | Window System<br>MOTIF *<br>X-Window * | Database Management<br>Sybase Server/Client<br>Ada Database Library | Communication Service<br>GOSIP (Future) *<br>TCP/IP *<br>BIND/HESIOD<br>KERBEROS<br>NFS * |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Hardware Layer | Standard Workstation (DECStation 3100) | Dedicated Processor (DECStation 5000) | Database Server (VAX 6420) | Communication Hardware | AMHS Hardware (Sun) | Video Hardware (Macintosh, other) | Map Graphics Hardware (Macintosh, DECStation) | 125 Identified Stove Pipe/ Comm Systems |

* Military or Commercial Standard

▨ Commercial-Off-The-Shelf

Figure 3. EUCOM Project Layered Software And Standards

validate user requirements while delivering capability to the user incrementally and reducing the document and review process overhead. The rapid prototyping methodology is somewhat similar to the early stages of the Spiral Model methodology. In fact, rapid prototyping is envisioned to play a significant role in the evolutionary development methodologies. Unfortunately, rapid prototyping has resulted in the development of very useful "throw away code" because of a lack of robustness, maintainability and expandability. Overcoming this deficiency in rapid prototyping is the primary basis of the RDM. Therefore, the objectives of the RDM are to preserve the flexibility and responsiveness of rapid prototyping and to deliver software meeting the requirements of good design and accepted coding standards, resulting in a maintainable system.

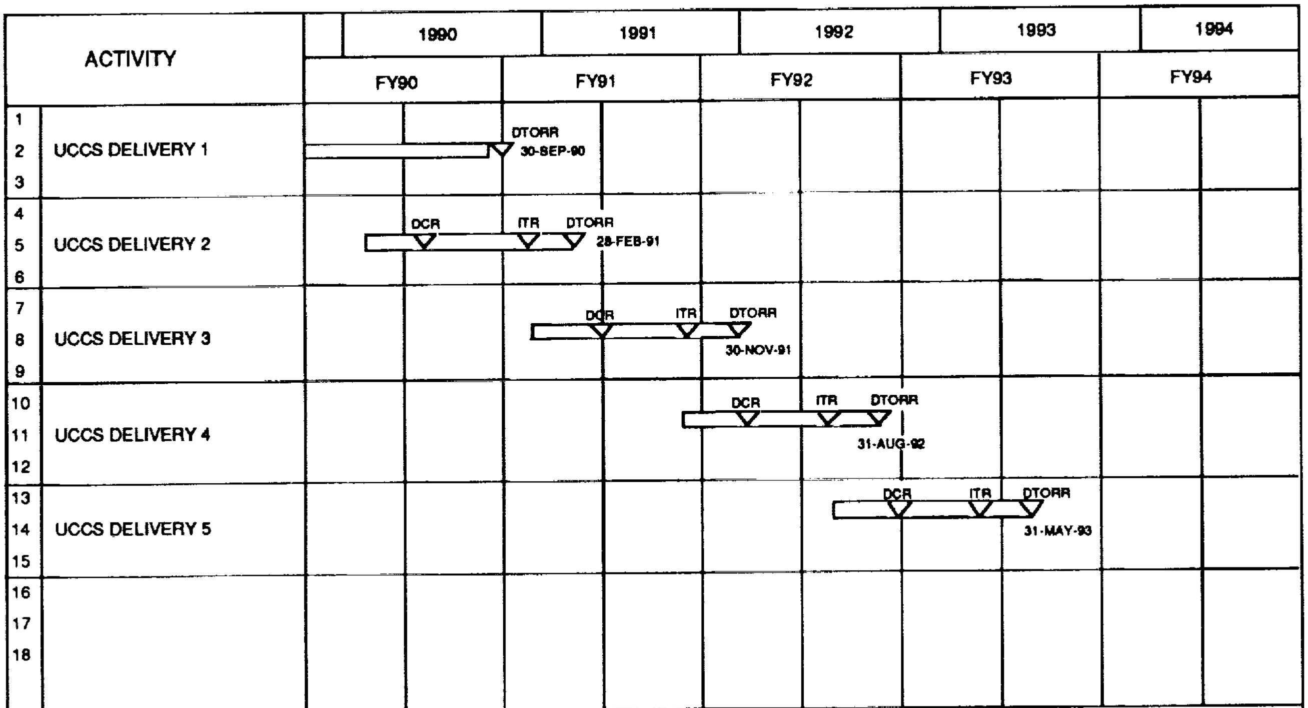
While RDM is not rapid prototyping with some added documentation; it is also not repetitive rapid executions of the Waterfall methodology, as in the Incremental Waterfall methodology. It is the iterative nature of the RDM with increasing documentation and formal reviews with each delivery that permits a departure from the Waterfall methodology and yet results in the end with a product identical to that developed using the Waterfall approach, from a maintainability perspective.

The RDM is a software development methodology and project management approach with a specific set of tenets. The basic tenets of the RDM that have been recognized to date are the following:

1. The system's functionality can be delivered and used incrementally. Each delivery must consist of a full system and not just pieces of the final system. Subsequent deliveries enhance or add functionality to previous deliveries.
2. The operational use of each incremental delivery provides active feedback of requirements into future incremental deliveries.
3. Users of the system will interact with developers extensively during the development of each incremental delivery. Users are active participants in all phases of the system development.
4. Users must agree that deficiencies in one delivery can be fixed in future deliveries.
5. The development cycle and the documents become progressively more formal as incremental deliveries are made.
6. The system is developed from project inception with an overall architecture that is sufficiently modular to allow for an evolving system development.

As currently budgeted, the EUCOM Project will proceed with five incremental deliveries resulting in the Early Operational Capability (EOC) at the end of FY93. The current schedule of deliveries is shown in Figure 4. One incremental delivery overlaps with the preceding and the succeeding deliveries in order to balance the staff and to achieve a 9 to 12 month interval between deliveries. Maintaining a nearly level staffing profile is necessary in order to achieve multiple incremental deliveries. Additionally, through successive deliveries, the users gain increasing understanding of computer automation benefits, which allow them to better articulate requirements and the developers gain more experience and expertise with the development environment. It is the refinement of requirements and increasing developer expertise through successive deliveries that permit greater detail of documentation and greater formality.

Figure 5 depicts the RDM process. The Project starts with an initial phase to establish an overall framework. This includes establishment of the overall requirements, schedule, budget and conceptual architecture/design. Each delivery starts with a planning and requirements phase. An agreement is reached with the Sponsor and the user on the functional requirements, schedule and budget for the delivery. Segment (sub-system) level partitioning, requirements definition and top level designs are then prepared. Also, detailed bottoms up costing and scheduling are performed. This results in a formal commitment review within JPL and a detailed commitment to the Sponsor/User. Implementation then proceeds similar to that of the Waterfall implementation phases, broken down into design, code and unit test and system test sub-phases. However, testing of the incremental system is somewhat different from the Waterfall approach, in that operational usability of the system is emphasized. This contrasts with an extensive acceptance test period emphasizing satisfaction of system requirements. Prior to installation and integration at the users' site, a formal Delivery Pre-Integration Review is held within JPL.



NOTES:

| | |
|---|---|
| DCR | = DELIVERY COMMITMENT REVIEW |
| ITR | = INTERNAL TRANSFER REVIEW |
| DTORR | = DELIVERY TO OPERATIONS READINESS REVIEW |
| UCCS | = U.S. COMMANDER-IN-CHIEF COMMAND CENTER SYSTEM |

Figure 4. EUCOM Command Center Project Schedule

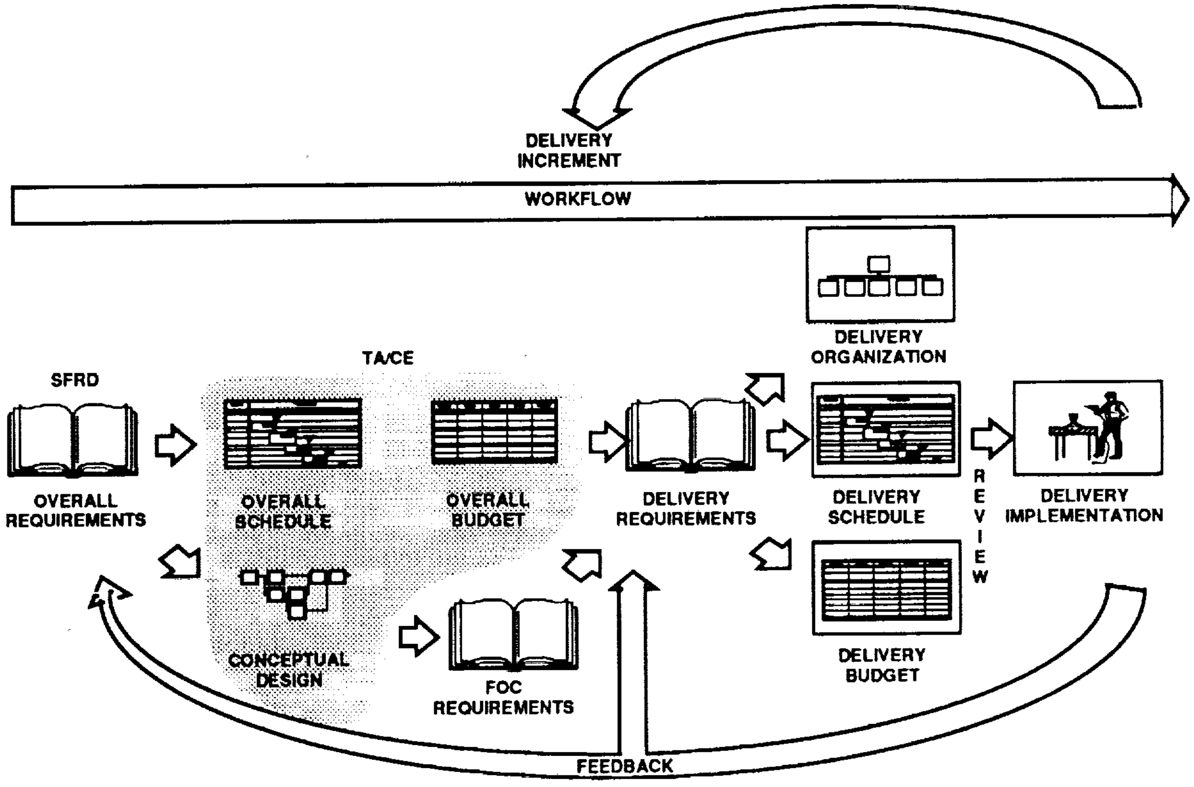


Figure 5. Rapid Development Process

As previously indicated, the users are involved extensively in the development of the system. The users state the requirements, through feedback of previous deliveries and discussions about future deliveries, at the beginning of each delivery. The users review all documents including System and Software Requirements Specifications and Software Design Documents. The users are part of the planning process setting priorities of requirements for each delivery cycle. The users review the implementation while it is in progress by frequent interaction with the developers (i.e. review of user interface storyboards and other design material, user interface prototypes, and partially working code). The users participate and make the final decision regarding cost/capability tradeoffs. The users are the final testers of the system as system operators/abusers. The users, through an established change request process, change the requirements for future incremental deliveries based on their operational experience. Finally, the users accept (or reject) the system. Since user participation is an integral part of the RDM, the user assumes "ownership" of the system early in the process.

The increased user participation is beneficial only if the users understand that deficiencies in one delivery will be addressed either as maintenance updates to the current delivery or included in future deliveries. On-site sustaining engineering support and timely and reliable developer-to-user feedback are provided to maintain a good working relationship with the user community.

## THE Ada PROGRAMMING LANGUAGE

The Ada programming language was developed in response to a perception on the part of the United States Department of Defense of a software *crisis* within the department's large scale complex computer software developments [6]. These complex software developments were perceived to lack maintainability, transportability to other hardware platforms, and lack support for code reuse. They were also difficult to integrate and manage due to a proliferation of source languages.

Ada was developed to address problems associated with large scale computer system developments and ultimately reducing total program lifecycle costs. In 1984, the DoD mandated that all future mission-critical software developments be developed in the Ada programming language [6]. The objectives of the Ada programming language were to: 1) develop a single, common programming language for all large scale software developments, complete with software development tools, to aid in all aspects of Ada software development, thus increasing the experience base of software programmers, 2) embody and enforce modern software engineering goals, such as program modifiability, efficiency, reliability and understandability, and 3) develop one language which would facilitate the transfer of software to different hardware platforms and operating systems by isolating the hardware dependent features of the language [6].

The Ada programming language was designed to embody and enforce modern software engineering principles such as data abstraction, information hiding, program modularity, localization, uniformity, completeness and comfirmability [6].

## THE ECCP SOFTWARE DEVELOPMENT LIFECYCLE

The EUCOM Project planners chose Ada as the standard programming language for all Project software development primarily due to the DoD Ada mandate and for the hope of realizing the benefits described briefly in the above paragraphs. However, the use of Ada required some modifications to the implementation phase definition of the RDM in the areas of sub-phase scheduling and costing, development tools implementation, and Ada coding standards.

### Implementation Phase Scheduling and Costing

The EUCOM Project uses a design-code-test ratio of 45-25-30. That is 45 percent of the Project's implementation phase is spent on preliminary and detailed design activities, 25 percent of the development schedule is spent on code and unit test, and the remaining 30 percent of schedule is spent on integration and requirements testing of the software (not including user tests as required for the RDM).

A higher percentage of implementation schedule is spent during the design phase than the normal 19 percent suggested by B. Boehm [ref. 1 pg. 65 for 32 KDSI] due to additional effort required to define all data types and develop package specifications. (In Ada, a collection of like procedures and functions is encapsulated into the Ada "Package". Each package consists of the package specification, which is the visible part of the package, and the body, which is the actual code.) Ada is a strongly typed language, and thus requires more time to define and negotiate all required data structures and type definitions.

Less time is required for the ECCP coding phase than the normal 55 percent as suggested by B. Boehm, because most sub-program interfaces are negotiated and agreed to during the design phase.

The test phase is different than traditional developments in that very little test time is used to find data type errors and sub-program interface mismatches, found during the design and coding phases. Also, recall one tenet of the RDM is that the user participates in all phases of the development, including the test phase.

In Ada, programming elements developed by one individual programmer, but used by others, such as procedure argument lists, data types, and external package references, are fixed in the package specification. For a large scale software development, this rigid definition of software program elements speeds usage of those elements by other members of the development team and speeds testing of sub-program interfaces.

Software Development Tools

As part of the ECCP's planning, a Rational Ada Development System was purchased from the Rational Corporation. The Rational is a complete Ada development environment with a 2167A document generator, an extensive configuration management system, a sophisticated language sensitive editor and Ada compiler. Code is first generated and compiled on the Rational. When the developer has achieved a successful compilation, the source code is transferred to the target environment (the DEC Station 3100 with Ultrix version 4.1) for unit testing. This transfer is managed automatically by the Rational.

The Rational development system, being dedicated to development of Ada software, has many features which support Ada programming that save a great deal of development time. One of the routine operations that a programmer performs during the coding phase is to refer to data type definitions. Although this may sound simple enough, with a strongly typed language, this is often time consuming and fatiguing. Most data type definitions are actually compound data types, with each component of the compound type itself being of a specific type. While coding, it is frequently necessary to examine several data type definitions at once, and these are commonly defined in different packages. The Rational system provides many shortcuts for viewing these definitions, saving a substantial amount of programmer time.

The Rational keeps track of highly detailed information regarding the interdependence of procedures, type definitions, and packages. This feature can be used to quickly identify, in a very specific way, any program entities which depend on another entity. The programmer uses this capability to determine the impact of any proposed changes, and to quickly traverse between the parts of the affected modules, speeding the process of making the software consistent with the modified portion.

Some other features of the Rational include syntactic assistance, which provides on-line Ada statement syntax completion and/or validation, semantic assistance which provides procedure argument completion and/or validation, incremental compilation which greatly reduces compile times of complex software programs, and an integrated configuration management system which provides automatic version and configuration management as part of the code generating process.

Since the RDM is based on multiple deliveries to the sponsor, feedback from the user is provided during the requirements definition and implementation phases of the next delivery. This feedback comes in the form of problem reports and requests for functionality enhancements. Because this feedback must be addressed during the development of the next delivery, configuration management plays an extremely important role in all phases of each delivery, regardless of the size of the programming staff, or number of lines of code to be managed. It is this overlapping of delivery developments and continual user requests for software fixes and enhancements that necessitate a great need for configuration management. For example, between the ECCP's last and current deliveries, the main database structure has changed to one that more adequately reflects real-world situations. But the previous delivery must still be supported at the same time.

The ECCP software configuration management process is well-defined and strictly enforced. Without this strong configuration management presence, much time would be lost to version mismatches and the resulting recompilation times. No time has been lost to date due to a software build with an incorrect package version.

Project Ada Coding Standards

The EUCOM Project has developed a set of coding standards which are tailored to the lifecycle discussed in this paper and take advantage of the benefits of the Ada programming language. Some examples of the Project's Ada coding standards are strong enforcement of Ada typing restrictions, utilization of Ada's run-time checking of type ranges, development of an error handling policy which takes advantage of Ada standard exception handling for calling applications but allows simple returned statuses from the system support layers to the calling application, and use of Ada generics and common libraries to facilitate code reuse and program modularity throughout the Project.

The ECCP Ada coding standards contain naming conventions to ensure that package and procedure names are meaningful and restrict the usage of the "use" clause, requiring the explicit use of reference labels. Both of these standards are aimed at providing more readable software.

In addition, all user interface routines are contained in standard libraries, and all applications are required to use these libraries. This enforces the Project's Man-Machine Interface policy at the same time.

The ECCP's coding standards facilitate good software engineering practices, which is required by the RDM for code expandability and modifiability from one delivery to the next.

## BENEFITS AND CHALLENGES OF ECCP SOFTWARE DEVELOPMENT LIFECYCLE

### Benefits

The benefits of RDM itself are many. Budgets and requirements for each increment can be fixed, thereby insulating the increment from the effects of overall funding and requirements volatility. The methodology is responsive to funding gyrations and requirements changes as a result of the evolution of the users' understanding of the system and the associated procedures for using it. The users gain a satisfaction with the system because of their "ownership" through participation in the development. Further, the RDM incrementally introduces changes in the operations organization rather than revolutionizing it with the introduction of a complete new system. Finally, RDM delivers capability faster than the turnover of user personnel - a major benefit for military systems.

One goal of the implementation phase of the RDM is to provide any reduction in time expenditure possible throughout each of the sub-phases. The ECCP has been able to provide these time savings through the use of the Ada programming language and its support tools. Time savings have been realized in the areas of software configuration management, using Ada to capture the design early in the design phase, making use of the language's extensive compile and run-time checks and using the language's inherent support for program modularity. Each of these areas is discussed below.

For the ECCP, each delivery increment was anticipated to be relatively small. Less than seventy thousand lines of code were developed for each increment by 12 to 14 actual software developers. The management of software change over multiple deliveries and an ever-changing and increasing line of code count increase the complexity of the software development environment. By using the Ada programming language and enforcing rigid configuration management standards, the difficult process of managing change to past and present software deliveries is minimized, almost to the point of being void of problems. In the accelerated schedule demanded by the RDM and with minimal resources available for configuration management functions, the combination of a strong configuration management policy, use of automated tools such as the Rational and the modularity features of the language all combine to save substantial programmer time, when compared with less automated or manual tools and the use of other languages lacking built-in modularity support.

The ECCP has designed its development process to exploit the advantages of the Ada language during each phase of the development. For the design phase, emphasis was placed on developing Ada package specifications, containing actual Ada code, rather than pseudo-code. By moving directly to compilable Ada statements, the step of translating pseudo-code into High Level language (HOL) statements was avoided. We believe that the inherent readability of Ada makes skipping the pseudo-code step possible.

The benefits of using Ada in place of pseudo-code are greater than just saving the time it takes to translate pseudo-code to HOL statements. By using Ada, the interdependencies of the compilation units are rigorously defined. Therefore, it is possible in the early design phase to determine whether there are any undesirable dependencies among the compilation units, and to resolve them in this early phase of development. By clearly and rigorously defining the architecture with actual Ada package specifications, architectural problems with the design can be identified and corrected before full implementation begins. This reduces the likelihood that significant portions of code will need to be rewritten without incurring the additional time for pseudo-code development.

During the actual implementation phase, Ada provides extensive error checking and error messages when compiling. Since procedures and functions have rigorously defined argument lists, and the arguments themselves are strongly typed, Ada can identify many common coding mistakes, and notify the programmer at the time of compilation. This provides some cost savings of time over the more traditional cycle of compiling, building, testing, and debugging without these compile and run-time checks, built into the language. This savings is especially important for software developed on a foreign host, such as the Rational system, and then downloaded to a target system. For such systems, the code generally needs to be compiled on both the development and the target system, in addition to the extra step of moving the source files from the host to the target. The sooner the programmer is aware of a problem, the more time is saved.

Another benefit of this approach is that the EUCOM Project is able to field versions of the system every nine to twelve months with each delivery providing significant new and modified capabilities to the operational environment. With the first decision support delivery in March of 1991, many initial requirements have undergone revision and redefinition based on the user's experience with the operational system. The Project's strong bias toward program modularity and its extensive use of common libraries permit significant code reuse from delivery to delivery. While some applications have been extensively reworked, the common libraries have been only enhanced and not re-designed. This has allowed us to provide significant new functionality in the next delivery, now scheduled for March, 1992 and at the same time making extensive modifications to some existing applications.

## Challenges

While early, incremental operational capabilities are well received by the user communities, our experience has shown that government review agencies are reluctant to accept the methodology and its initially less rigorous testing and review processes. Our challenge has been to allow close participation with government Quality Assurance organizations, which tend to require complete documentation sets and lengthy review cycles, while still maintaining the accelerated delivery schedule dictated by the methodology.

One challenging aspect in the selection of Ada has been that Ada requires total control of the program execution environment. By controlling the program execution environment, all the run-time benefits designed into the programming language can be realized. Unfortunately, this philosophy conflicts with other operating environments which tend to control the environment. Examples of conflicting controlling environments are the UNIX operating system and the X Window Graphical User Interface. For example, we have demonstrated that the use of multiple Ada tasks in a single UNIX process have yielded unpredictable and erroneous results in our target environment. Careful management of the interaction between each of these environments is required for a successful implementation of these government standards. Our solution to the problem has been to restrict the use of Ada tasking by making each Ada task a separate Unix process.

One pitfall associated with this approach is the increased cost of supporting the operational system due to its early fielding. The operational support of the system has been addressed through a combination of on-site and military support teams. USEUCOM has provided a 24 hour operations support team during the first delivery in addition to the on-site JPL support team. The JPL on-site support team provides assistance for technically challenging problems, not day-to-day operational problems. Additionally, USEUCOM is developing an on-site user support team, responsible for day-to-day user training, and general user support. The Command and JPL have accepted this cost as necessary for the continued support of the user community.

Another pitfall associated with this approach is the possibility of architectural changes due to future user requirements which were not anticipated during the original architectural design. This has been addressed by the modularity of the system architecture, minimizing and isolating the impact of future architectural changes. Prior to each commitment review, architectural changes are factored into the committed work plans and agreed upon by the sponsor/user.

## RESULTS

The ECCP is midway through development of the key elements of a modem command and control system, adhering to government-mandated (and sometimes conflicting) software standards. The Project's alternative methodology has the advantage of delivering substantial capabilities to the user much earlier than other methodologies without sacrificing lifecycle development quality or rigor. During each delivery, the duration allocated for the code and test phases have been reduced allowing more capability to be developed and delivered.

The Project has developed portable software modules which can be applied to any distributed computing environment requiring monitor and control, client-server database driven decision support applications in a command and control atmosphere. The use of standards, development of reusable software "modules" and the early delivery of an operational system has proven to be very beneficial in the current limited budget environment.

## ACKNOWLEDGEMENT

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

## REFERENCES

1. Boehm, B., Software Engineering Economics, (Englewood Cliffs, NJ: Prentice-Hall, 1981).

2. Firth, R., Wood, B., Pethia R., Roberts, L., Mosley, V., Doice, T., "A Classification Scheme for Software Development Methods," Technical Report, Software Engineering Institute, Carnegie-Mellon University (November 1987).

3. Scacci, W., "Models of Software Evolution: Life Cycle and Process," SEI Curriculum Module SEI-CM-10-1,0 (October 1987).

4. Sodhi, J., "Managing Ada Projects," (Tab Books, 1990).

5. "JPL Software Management Standards Package" the Jet Propulsion Laboratory, D-4000 (Internal Document) Version 3.0 (December 1988).

6. Grady Booch, "Software Engineering with Ada," (Benjamin/Cummings, 1986).

7. Military Standard, "Defense System Software Development, DoD-STD-2167A," (Washington D.C., June 1988).