AFATDS EXPERIENCE

presented at the

Fifth Washington Ada Symposium
(WADAS'88)
June 29, 1988

by

Donald G. Firesmith

Magnavox Electronic Systems Company
M/S 10-C-3 Dept. 566
1313 Production Road
Fort Wayne, IN 46808
(219) 429-4327
firesmit@ajpo.sei.cmu.edu

1) AFATDS Overview

- The Advanced Field Artillery Tactical Data System (AFATDS) is a very large automated command and control system designed for the US Army to support all levels of command from platoon to corps.

- AFATDS implements the full range of field artillery command and control functions as well as fire support, control, and coordination for cannon, rocket, missile artillery, mortars, air support, and naval gunfire.

- The system requirements vary from soft real-time functions to totally administrative functions.

- AFATDS employs an extremely high degree of automation that features advanced decision support tools, information presentation techniques, and concurrent foreground and background operations, etc.

- Contractor:

  Magnavox Electronic Systems Company

- Contact person:

  Mr. Harold (Skip) B. Carstensen,
  AFATDS Software Director,
  (219) 429-5272

- Contract award: May 1984

- AFATDS is a multi-phase project nearing completion of Concept Evaluation Phase (CEP).

- The main software development method was Object-Oriented Development (OOD).

- The general software development philosophy was:

  1. Follow OOD to implement the required capability
     (Get it working).

  2. If necessary, increase object code speed
     (Make it fast enough).

  3. If necessary, reduce the size of the object code
     (Make it small enough).

  If possible, rely on unit-level redesign (to fix obvious inefficiencies) and compiler optimizations to increase speed and decrease size. Do not hand-optimize code for the project compiler unless absolutely necessary and then only if the software is on a critical performance path.

- Host development system:
  - DEC 8800, 8650, 8600, and 11/780 with VAX/VMS
  - DEC Ada compiler and tools
- Target system:
  - Motorola MC68020-based (32 bit) workstation with touch-entry graphics display.
  - Telesoft TeleGen2
- Beta test site for:
  - Motorola M68020 and MMU chips
  - TI 802.5 token ring LAN chip
  - DEC Ada compiler and related tools
  - Telesoft TeleGen2 Ada VAX and 68K Cross Compiler and related tools
  - OOD
  - Ada and Ada PDL

- Training was primarily provided by EVB Software Engineering.

- The training was of high quality, but expensive and difficult to tailor for consistency with company standards, etc.

- AFATDS was Magnavox's first significant Ada project and first large software system project.

- "The AFATDS program ... is a sound technical success for Ada. ... Magnavox, the AFATDS contractor, has been able to use Ada throughout the design and development phases to support its software engineering methods. ... [However,] the development team has been brought to a total standstill on dozens of occasions by reviewers, [GAO] investigators, and IV&V (Independent Verification and Validation) contractors."
  Ralph E. Crafts, Editor
  Ada Strategies, March 1988

- AFATDS consists of a set of 7 major hardware components and the following 9 software configuration items:

  - Execution Environment:

    Operating System
    Data Management
    Information Management
    Communications Support
    Communications Interface

  - Applications Software:

    Fire Support Planning
    Fire Support Execution
    Movement Control
    Common Functions

- With the exception of less than 3 KSLOC of operating system and communication software, AFATDS was written completely in Ada.

- AFATDS can be configured from a single hardware component to a large center contained in several tactical vehicles.

AFATDS SOFTWARE SIZE

REL.	DATE	FILES	LINES	NCNBs	TSCs
4.04	25 FEB 88	7,553	2,517,594	1,175,498	409,982
4.03	16 DEC 87	7,428	2,495,749	1,160,332	401,933
4.02	03 SEP 87	6,691	2,320,000	1,061,487	370,469
4.01	MAY 87	5,109	1,806,101	819,792	278,085
3.0	JUL 86	?	?	494,000	?
2.0	APR 86	?	?	254,000	?
1.0	FEB 86	?	?	51,000	?

LINES = PHYSICAL LINES

NCNBs = NON-COMMENT, NON-BLANK PHYSICAL LINES

TSCs = TERMINAL SEMICOLONS (DETERMINED BY COUNTING ALL SEMICOLONS EXCEPT THOSE IN COMMENTS, QUOTED STRINGS, AND FORMAL PARAMETER LISTS -- NOTE: THIS IS NOT THE SAME AS ADA STATEMENTS)

AFATDS PRODUCTIVITY (NCNB SLOC)

	RELEASE	1	2	3	4.01	4.02	4.03	4.04	TOTAL
△ SIZE	EXEC. ENV.	?	?	?	?	20,907	-4,129	2,752	339,162
	APPLICATIONS	?	?	?	?	251,922	102,974	11,881	789,775
	AFATDS TOTAL	51,000	203,000	240,000	248,630	272,829	98,845	14,633	1,128,937 *
DAYS	EXEC. ENV.	N/A	N/A	N/A	4,245	1,730	1,136	588	13,769
	APPLICATIONS	N/A	N/A	N/A	7,019	5,307	3,179	1,589	24,461
	OTHERS	N/A	N/A	N/A	1,291	733	766	392	5,169
	AFATDS TOTAL	6,116	3,983	5,325	12,555	7,770	5,082	2,569	43,400
SLOC ---- DAY	EXEC. ENV.	?	?	?	?	12	-4	5	25
	APPLICATIONS	?	?	?	?	47	32	7	32
	AFATDS TOTAL	8	51	45	20	35	19	6	26

EXECUTION ENVIRONMENT =

OPERATING SYSTEM	(52 KSLOC)
DATA MANAGMENT	(88 KSLOC) *
INFORMATION MANAGEMENT	(140 KSLOC)
COMMUNICATIONS SUPPORT	(52 KSLOC)
COMMUNICATIONS INTERFACE	(44 KSLOC)
TOTAL	(376 KSLOC)

* INCLUDES 36.5 KSLOC SUBCONTRACTED

APPLICATIONS SOFTWARE =

FIRE SUPORT PLANNING	(276 KSLOC)
FIRE SUPPORT EXECUTION	(254 KSLOC)
MOVEMENT CONTROL	(48 KSLOC)
COMMON FUNCTIONS	(212 KSLOC)
TOTAL	(790 KSLOC)

OTHERS =

- SOFTWARE MANAGEMENT
- SOFTWARE TEST
- RELEASE INTEGRATION AND TEST
- SOFTWARE QUALITY ASSURANCE
- DATA ITEM SECURITY SCHEME
- SCREEN DESIGN SUPPORT

REUSABLE SOFTWARE NOT DEVELOPED ON THE PROJECT (I.E., GRACE, MATH PACKAGES) WAS NOT COUNTED!

REUSABLE SOFTWARE DEVELOPED ON THE PROJECT WAS ONLY COUNTED ONCE!

DAYS = 8 HOURS BILLED

SOURCE = MS. KAREN SIDLEY (DETAILED PAPER IN PROGRESS)

2) AFATDS Lessons Learned

- "Ada is being successfully used today in military programs, such as AFATDS." Report of the Defense Science Board Task Force on Military Software, September 1987

- Management support for Ada is essential. Innovative software development methods are less likely to be implemented without active management support at all levels.

- Low and mid-level technical managers must remain technically current on projects involving a new software development method, a new language, and a new mindset. They must not become bogged down in micro-scheduling and status reporting.

- Software engineers must be an integral part of the system engineering effort.

- Plan on spending more effort on requirements analysis and less effort on integration/testing than is typical on projects using classical methods and the "waterfall" life-cycle. According to Skip Carstensen, effort on a large Ada project can be allocated as follows: 55% to requirements analysis and design, 10% to coding, and 35% to testing and integration.

- Increased government/user participation benefits the project.

- Pertinent government and IV&V personnel must receive training in software engineering, the project software development method (if state-of-the-art), and Ada.

- Spend adequate time and money prior to project initiation to:

  1. Determine the appropriate software development method.

  2. Develop software standards and procedures (e.g., Ada coding standards).

  3. Train technical management and the developers in software engineering, the project software development method, the software development environment, and Ada.

- Because ongoing training in the project software development method and Ada (e.g., via Help Desks) is very cost-effective, designate adequate personnel to provide such expertise.

- Use an Ada-oriented software development method (e.g., OOD).

- OOD is not compatible with:

  1. Functional decomposition methods.

  2. Obsolete DoD development standards (e.g., DOD-STD-2167 and MIL-STD-1521) based on the classic "waterfall" life-cycle.

- OOD naturally produces a very large number (over 90%) of very small and simple Ada programming units (McCabe's metric less than 4).

- Such units cause far fewer problems during integration and test.

- With the use of Ada for design as well as coding, PDL's as such are no longer needed.

- Ada is not compatible with the DIDs associated with certain DoD standards (e.g., DOD-STD-2167, MIL-STD-483, and MIL-STD-490). Attempting to conform to such standards in a rigorous manner on a modern Ada project forces numerous convolutions and wastes a great deal of time. Such standards are obsolete and do not lend themselves to being adapted to modern software engineering practices. For example, one must document the inputs and outputs of the standard Ada "Calendar_IO" and "Text_IO" packages because these standards require the description in detail of all unit inputs and outputs.

- MIL-STD-483 and MIL-STD-490:

  1. Do not support either modern software engineering or OOD.

  2. Greatly decrease developer productivity without improving software quality.

- Proper (i.e., Ada- and method-oriented) Software Development Files (SDFs) are better than traditional C5s for documenting Ada designs.

- Evaluate each compiler vendor's commitment to supporting your project. How often will updates be provided? Will your input affect the vendor's priority with regard to fixes? How easy will it be to report problems, get responses? What will the vendor's response time be on getting a fixed product?

- Test and benchmark compilers, etc. prior to making a decision.

- Do not base your decisions on the verbal promise of a vendor.

- Designate someone to act as the contact person dealing with (and the expert on) compilers, tools, and their vendors.

- The lack of user friendly automated tools can significantly increase cost and schedule.

- Tools must support the project software development method.

- The use of beta site versions of immature compilers required more computing power and disk space than we imagined possible. For example, we needed fourteen 400 megabyte disks.

- The pragma INTERFACE feature of the Telesoft TeleGen2 compiler works extremely well with small assembly routines.

- Due to compiler immaturity, certain Ada constructs and procedures may be rewritten to provide the same capability with significantly less memory required. Style changes resulted in an overall reduction of 20 to 50 percent in object code size of certain units. Note that this is extremely compiler-dependent.

- Compiler optimization is critical. The use of an optimizer should provide at least 10 to 30 percent object code size reduction. One should be able to reduce the object code size of imported (i.e., "withed") packages another 5 to 15 percent with a smart linker. Note that this is also very compiler- and time-dependent.

- One of the most difficult management jobs when dealing with the Ada technology transition is achieving a cost-effective balance between sufficient:
  - Control through the use of sound engineering methods.
  - Flexibility to immediately adapt and apply the numerous lessons learned.

- Significant reuse is not only practical, but also necessary if one is to meet cost and schedule constraints on a very large project.

- AFATDS has proved the practicality of Ada reuse. The reuse of Ada software on AFATDS meant that the developers did not have to develop over 100,000 lines of deliverable code. Approximately 13 percent of the total delivered software was reused software for a savings of slightly under 190 man months or 2 calendar months off the overall project schedule.

- There are significant places in a major project where very similar functionality is being performed. Major portions of software developed can be reused with only minimal modification or addition.

- Based on our experience, the use of OOD made reuse easier.

- In order to make maximum use of the reusability that OOD provides, one would have to use OOD from the very beginning of the project. Because the DoD is entrenched in functional decomposition, one may be forced to wait until the CSCI level to initiate OOD.

- There is a strong tendency for developers to think that they can develop better software than that in the reuse libraries. The real reason for this is often considerably more psychological than technical due to developer pride and fear of the unknown.

- We can easily increase the amount of software being reused to 25 percent on the next similar project, and 50 percent reuse on future projects appears possible.

3) Information Sources

- "Experiences in Achieving Size and Performance Requirements on a Large Ada Project",
  Harold B. Carstensen, Jr.,
  Computers in Aerospace Conference,
  June 1987

- "Experiences in Delivering a Large Ada Project",
  Harold B. Carstensen, Jr.,
  Military Computing Conference,
  5 May 1987

- "A Real Example of Reusing Ada Software",
  Harold B. Carstensen, Jr.,
  Fourth National Conference on Methodologies and Tools,
  2 March 1987

- "The Management Implications of the Recursive Nature of Object-Oriented Development",
  Donald G. Firesmith,
  AdaEXPO/SIGAda Conference,
  7-11 December 1987

- "Transitioning Developers to Ada",
  Donald G. Firesmith,
  Second Annual ASEET Symposium,
  9-11 June 1987

- "The ETAS Central Processor: A Case Study in Reusable Software",
  Richard A. Howard,
  Tactical Communications Conference - 88,
  3-5 May 1988

- "Experience and Lessons Learned in Transporting Ada Software",
  Karen E. Sivley,
  Joint Ada Conference,
  16-19 March 1987

- "Development Software Configuration and Integration in a Large Ada Project",
  David H. Ternes,
  SIGAda Conference,
  9-11 December 1987