IMPROVED HELLFIRE, SUCCESSFUL USE OF THE ADA LANGUAGE IN AN EMBEDDED, REAL-TIME APPLICATION

Bill Miller

Rockwell International
Missile Systems Division
Duluth, Georgia 30136

ABSTRACT

The Missile Systems Division of Rockwell International has been involved in the successful development of embedded, highly real-time, Ada software for controlling the guidance of the AGM-114 HELLFIRE antitank missile. This project is the first operational tactical missile application using Ada. This software development effort was a very typical embedded project, with many of the typical embedded project challenges. Using Ada for an embedded, real-time system provided Rockwell with a distinctive set of advantages and disadvantages. It was not only the use of Ada, but a total "Software Engineering" approach that resulted in the overall success of the software project.

1. IT CAN BE DONE!

There appears, in the military software industry, to be a significant amount of technical armwaving with regard to Ada. There are innumerable seminars, symposiums, papers presented, theories expounded and general beating of our collective software chests. The actual concrete accomplishments within the Ada world do not yet appear to measure up to our expectations.

However, the Missile Systems Division of Rockwell International has accomplished something unique while maintaining a very low profile. It has actually developed a working, embedded, tactical missile control software application in Ada. Not only has it accomplished this feat, a first for any DOD contractor; it has done all of this during the Full Scale Engineering Development of the digital autopilot upgrade to AGM-114 HELLFIRE modular missile program.

This program and Ada success was and could only have been accomplished with significant Ada training, planning, use of structured development techniques and total support by engineering and program management as well as an honest, frank, and cooperative relationship with the U.S. Army, the customer for the HELLFIRE missile.

The objective of this paper is to describe some of the experiences encountered, to review some of the methodology used and to make recommendations for future real-time, software projects that will be using Ada.

2. WHAT'S A HELLFIRE?

A HELLFIRE missile is a modular (launchable from various platforms), anti-tank missile. It is 7 inches in diameter, 5 feet long and weighs about 100 pounds. It is laser-guided and is supersonic.

The digital autopilot's job is to sample the seeker, gyros and fin positions, perform the necessary calculations and to output new fin positions to maintain a stable flight on the way to the target. For an object that travels at 700 feet per second this provides a significant processing challenge to the microprocessor and the software.

Although the entire Improved HELLFIRE project resulted in the development of a moderate amount of Ada code, the amount of software that actually resides in the missile is not particularly large: approximately 3500 lines of Ada and 2000 lines of assembly language. The total software is about 55 Kbytes of ROM memory with the Runtime Services Library (RSL) using about 12 Kbytes of the 55.

3. ONLY THE BEST WILL DO

When Missile Systems Division of Rockwell International in Duluth, Georgia began the pre-contract award effort for the Improved HELLFIRE we had absolutely no Ada experience. When we started the program, we hand-selected only the best software engineers (i.e. engineeers with specific software education) from within the Duluth division. The average software engineer had 6 years of embedded software experience. The company funded a three week intensive, off-site Ada training course taught by Data General here in the Atlanta area. The course consisted not only of Ada training, but Ada training using an actual MSD control system application. The course was attended by

every software engineer who was selected to work on the project as well as the software quality engineer who was selected for the project.

As a result of early identification and training of all engineers who were to work on the program, the project was fully staffed from the first day of the effort.

The software manager of the group (also hand selected) was an accomplished software engineer who also had significant engineering management experience.

4. PLANNING THE HARDWARE

As a part of the overall planning, every item of anticipated test equipment was identified and acquired. This list of equipment was crested by the most senior members of the development team over the period of a couple of weeks prior to the start of the software development effort. Their task was to attempt to identify the types of problems that could arise during the development effort and determine if a specific tool existed that could facilitate solving the problem. With very few exceptions the set of tools defined and acquired was exactly what was needed. This wise investment in the time to define the equipment and in capital equipment itself resulted in having the necessary tools at the appropriate time. Very little development time was lost due to running around trying to dig up some old piece of test gear.

Another planning effort centered upon how many digital autopilot boards (target hardware assets) would be available for the software development team. This author has seen too many projects where only one new board is allocated to all of engineering. It quickly becomes cobbled up with hardware patches, making it marginally useful for software check-out.

5. WHICH COMPILER?

The compiler selection for this program was tightly coupled to the microprocessor selection. Availability of processors as well as the availability of a validated compiler were both a concern. (We were, because of requirements constraints, only interested in microprocessor chips, not single board computers.)

At the time we were selecting the compiler, approximately summer of 1986, the processor/compiler pair that best offered the possibility of actually being available was the Intel 80286 processor and the Softech compiler.

We had narrowed the processor selection to a couple of different new processors, each processor had promised the imminent validation of a compiler targeted to their processor. The feeling was to go with the vendor that actually had working hardware. It was felt that if we went with the vendor that had working hardware and the Ada compiler failed, then we could write in some other language. However, if we went with someone that had a validated compiler but no hardware, then we could end up with software but no processor.

6. COMPILER EXPERIENCES

As an opening remark here, let me warn the reader that validation isn't everything its cracked up to be. Our first experiences with a "validated" compiler resulted in the compiler frequently failing in new and wondrous ways. It did everything from hanging up the host computer system (a VAX 11/785) to producing bad code and everything in between. (The bane of being an early adopter is that you are a guinea pig, attempting to do things that have never been done before.) One normally assumes that adding two variables and putting the result into a third variable does not require significant amounts of debugging. With any new compiler, this is not necessarily the case.

While solving these compiler-related problems can be technically exciting for awhile, eventually the engineers get tired of "debugging the compiler vendor's product" and you the manager get tired of reporting to your management that "we've looked at the code 5 different ways and there should be no reason why it shouldn't run." Each new compiler version release became a new uncharted land to be conquered. Eventually we became tired and pressed for time and stayed with a particular version of the compiler with all known bugs either solved or worked around. As a result of the above experiences, we have become somewhat wary of switching from a known good compiler version to a new "better" version, or to another vendors compiler.

However, we were not left to fend for ourselves at each new compiler version release. During the initial installation of any new compiler version, Softech has been extremely helpful in solving the problems that have been encountered. They apparently have a reasonably sized software staff and have assigned one person to be our interface and have worked quite closely with our software engineers. As a result, their mean time to solve each problem seems to be quite reasonable.

7. WALK BEFORE YOU RUN

We had, at the beginning of the project determined that it was not a goal to implement a sophisticated program from the Ada standpoint. Too often, I have seen the mistake made of using the project as a justification for experimenting with new tools. The sole goal of Improved HELLFIRE was to develop a program that would control the missile flight and to use the features of Ada that best facilitated this goal.

While Ada offers many language features that are highly attractive, initial tests, or trade studies, showed some of them to be far too costly with regard to time and/or memory.

The major language features that our HELLFIRE design avoids are: generics and tasking. Whereas these are two prime characteristics of the Ada language, the highly critical time constraints of our system prevented either of these features from being usable.

Task context switching, to be usable for us would need to occur in the 10 to 20 microsecond range. For generics there is a run-time instantiation tax that we felt was not worth the price. Instead of tasking we used a background/foreground technique to handle the two primary processing rates. We wrote our own rendezvous software to move data between the two processes.

The software design methodology use for this project consisted primarily of Yourdon™ Data Flow Diagrams. This method, although poo-pooed by many critics, has been used successfully by the author continually on small, highly realtime projects over the last 8 to 10 years. Whereas other, newer design methods such as OOD™ (Object Oriented Design) or PAMELA™ (Process Method for Embedded Large Systems) might have offered some advantages, it was decided to have only one new variable, Ada, in this project's equation.

In an attempt to accomplish some amount of early software prototyping, prior to creating code bodies, specifications for all software modules were created, compiled and linked with stubbed bodies to ensure that consistent data flow was occurring throughout the software system. Additionally, any assembly language that was used was interfaced to other Ada software via an Ada specification.

This early definition of acceptable language features was highly contributory to the software's overall cost and schedule successes. Redoing the software following a completed design, coding and check-out would not have allowed us to meet our schedule.

8. BEWARE OF RUN-TIME

When we first embarked upon our Ada effort we were absolutely unprepared for the many implications of the run-time. Our timer structure, interrupt system and memory layout were somewhat non-standard. This non-standard hardware didn't prove to be too great a problem with the Sof-tech Run-time. What proved to be too great a problem was getting all the run-time features that we needed without "buying the entire farm" of run-time (text I/O, tasking, etc.). Softech offers two versions of run-time: mini-RSL (Run Time Services Library) and full RSL. Unfortunately mini-RSL didn't have everything we needed and full-RSL was too big for us (taking all 64K of the memory allowed to be used by the Improved HELLFIRE software). As a result Softech created a midi-RSL for us (at a cost) that contained the necessary features.

The ideal run-time is one that would be fully configurable, allowing one to select and choose only those desired features.

Once the technical issues of run-time are solved, then one must climb the multiple copy licensing wall. For the HELLFIRE program, this may not be a large dollar item. However, the legal ramifications are far too mind-boggling for this Software Manager to comprehend. The HELLFIRE digital autopilot is to be dual sourced once it gets into production. Who needs to get licenses, the Army, both contractors or all three? What about if one of the contractors wants to build missiles for another arm of the armed services or for a foreign country? These are questions that still remain to be answered.

9. THROUGHPUT PROBLEMS!

Throughput was anticipated from the beginning of the program to be a bottleneck. As a result it was closely monitored beginning at the earliest design stage. The problem with throughput was twofold: We had the real physical limitation of the microprocessor, and we had the problem that 100% Ada would reequire customer concurrence and an approved deviation.

From the very beginning, there was a serious concern that 100% Ada would not fit into the available throughput of the microprocessor. However, until the entire system was built up and a system was put into hardware-in-the-loop testing, no real, total throughput measurement could be made. *(A frightening but common prospect was that we would not really have a firm handle on total throughput until month 20 of a 38 month project.)* Moreover the complexity of the requirements

made it difficult to predict software "hot spots". Therefore we decided to develop the entire system in Ada with the intention of converting to asssembly language later as required to achieve the necessary throughput margins.

As the requirements firmed up, a throughput budget for each module was established. If all module bugets could be achieved, and acceptable throughput margin would be attained. As testing progressed, any module that grossly missed it's budget was re-evaluted in a couple of ways. First the code was reviewed to determine if changes could be made to remove subroutine or function calls. Then the requirements were reviewed to determine if changes or simplifications could be made to remove subroutine or function calls. Then the requirements waere reviewed to determine if changes or simplifications could be made that would still accomplish the intended task but allow the module to remain as Ada code. Next, the code itself was reviewed to determine if coding any of the math as integer (since we used floating point types rather heavily) was feasible. Finally, after above efforts failed, modules were converted to assembly language. However, the original Ada code was retained, maintained and delivered as part of the final product. This allows for an easier growth back to Ada if faster processors become available. Modules with marginal timing were left unchanged until the entire system was put together and tested.

The resultant delivered code consisted of a mix of both Ada and asssembly, with the minimal amount of assembly used to achieve necessary throughput margins. There was defiantly a trade of throughput margin versus percent Ada that waas made with the Army's concurrence. In the particular case of Improved HELLFIRE, throughput is sitting at approximately 80% and the Ada/Assembly mix is sitting at about 80% Ada (versus the contract desired values of 60% throughput utilization and 100% Ada).

10. SOFTWARE QUALITY'S ROLE

As a military contractor, my largest complaint with Software Quality Assurance (SQA) has been that too often they act as Software Paperwork Quality Assurance. While it is important that paperwork have a minimum of errors, I believe that SQA can and should have a larger and more critical role in the development of software. This role requires SQA engineers who understand software systems instead of paperwork systems. While this concept may seem obvious at first, one needs to consider that development engineers want to do engineering, not quality.

For this program we had a software quality engineer assigned full time to this program. This SQA engineer sat with the development engineers, participated in design decisions reviewed design specifications and most importantly had access via the VAX to read all the engineering development files. The SQA engineer participated in the above mentioned Ada training course. As a result he was well prepared to and did spend a significant amount of time actually looking at PDL and source code during its development to ensure that not only were standards being adhered to but to also ensure that technical requirements are being satisfied.

Moreover, the sofware quality engineer ensured that test plans were followed and tests logs with results were maintained.

Finally, the software quality engineer was used as a developmental configuration management librarian. In this capacity he was able to ensure the integrity of tested software modules.

11. RECOMMENDATIONS

Ada was mandated on the Improved HELLFIRE program. Perhaps we at Missile Systems Division were too naive to take an exception to the Ada requirement at the beginning of the project. If we had Improved HELLFIRE would have been just another project with some software in it and I wouldn't be here today telling you what can be accomplished. Ada is really not a word that should scare or even challenge potential real-time software contractors. Ada is a tool for developing software that offers tremendous advantages over other High-Order Languages. As the products of the Ada compiler vendors become more mature and sophisticated, it is obvious to this writer that Ada will become the preferred development language of all software engineers.

This software project is not only a testimony to the Ada programming language but also to modern software engineering techniques. The keys to this successsful software program are embodied in the following points:

1. Adequate Planning
2. Anticipation of and tracking of potential problems
3. Early Prototyping
4. Sufficient capital equipment
5. Sufficient target hardware assets
6. Highly qualified and trained engineers
7. Intense SQA involvement
8. Use of Ada!

IMPROVED HELLFIRE, SUCCESSFUL USE OF THE ADA LANGUAGE IN AN EMBEDDED, REAL TIME APPLICATION.

by
Bill Miller
Manager, Software Design

Rockwell International
Missile Systems Division
Duluth, Georgia 30136

C001

Bill Miller has 16 years of real-time, commercial and military software development experience. He has been with Rockwell International for the past five years and is currently the Manager of Software Design at Missile Systems Division. Prior to this assignment Bill was the manager of Real-Time Software Engineering for the Strategic Defense and Electro-Optic Division at Rockwell, guiding the developement of multiprocessor, real-time image processing and signal processing applications. Bill has a B.S. in Computer Science from the University of California at Irvine and an MBA from West Coast University.

Improved HELLFIRE Goals

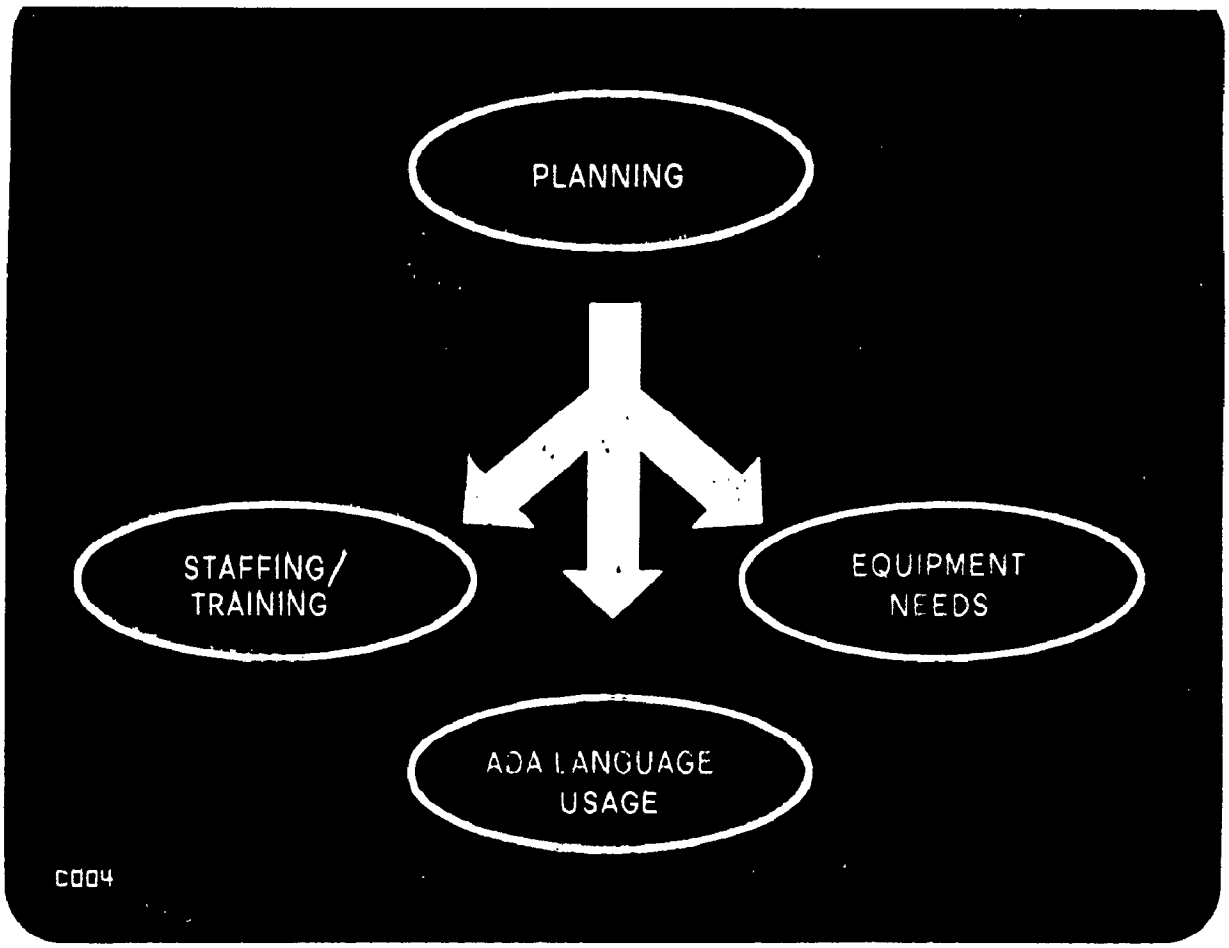
- -Develop Missile Control Software
- -Develop Software in Ada
- -Use no more than 50% of available memory
- -Use no more than 60% of available throughput
- -Develop Software within established schedule and budget
- -Create the "whole 9 yards" of software documentation

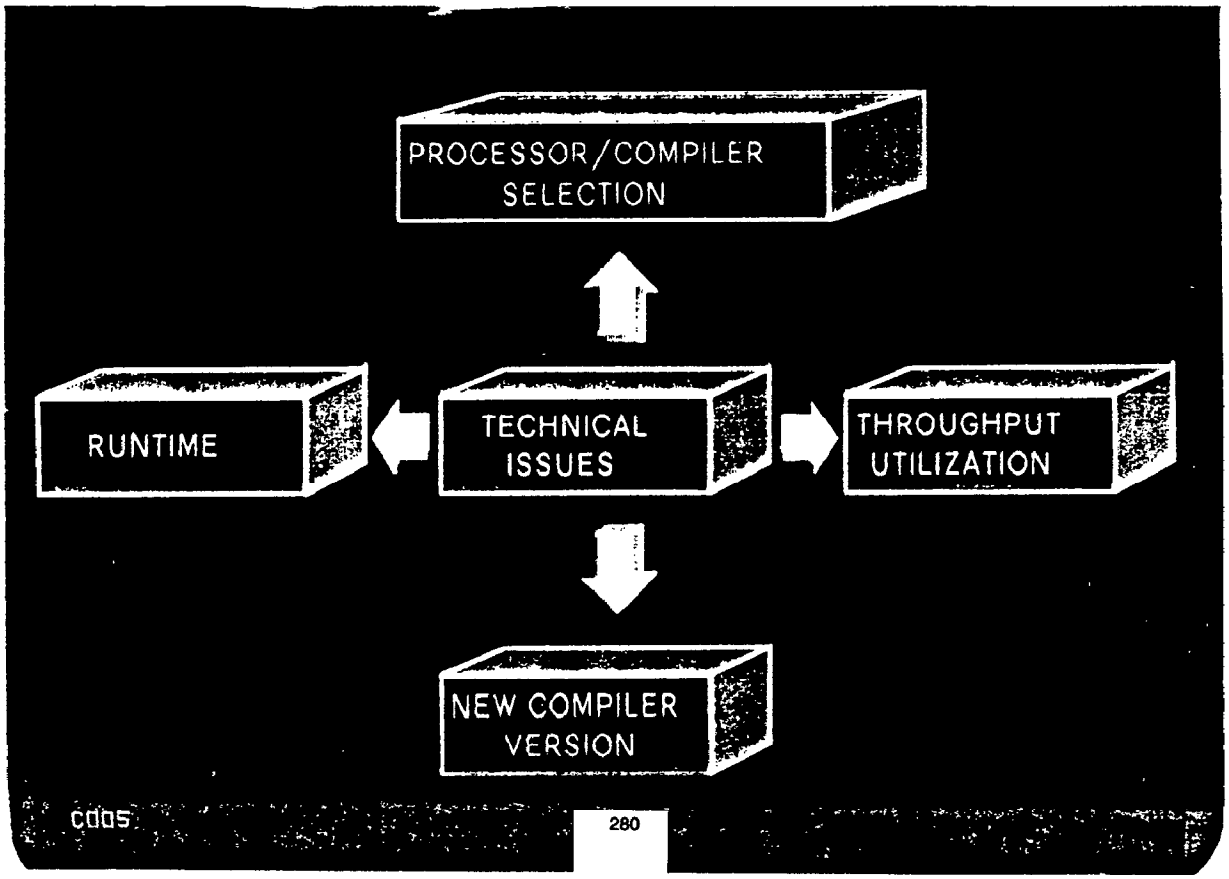
C002

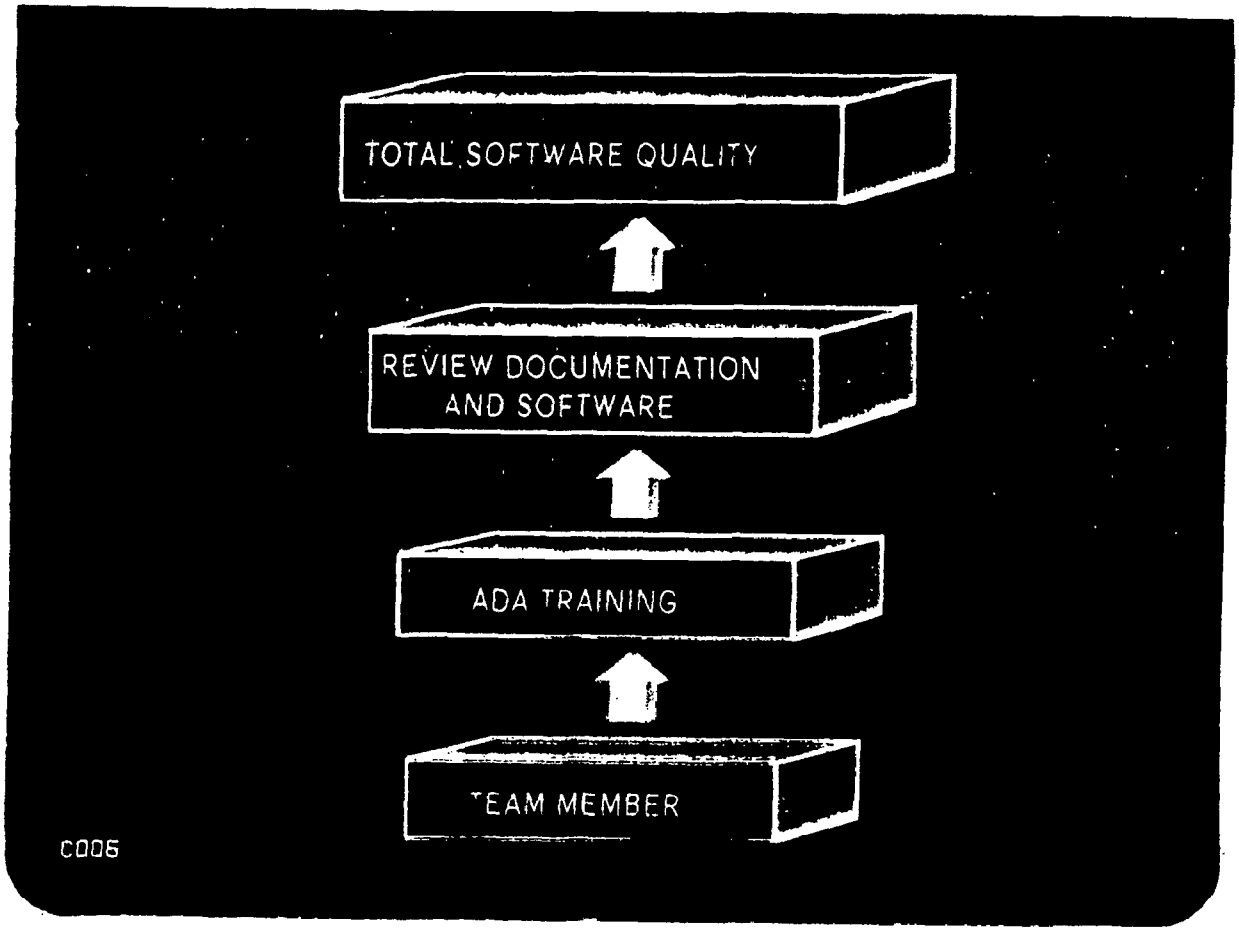
Improved HELLFIRE staffing consideration

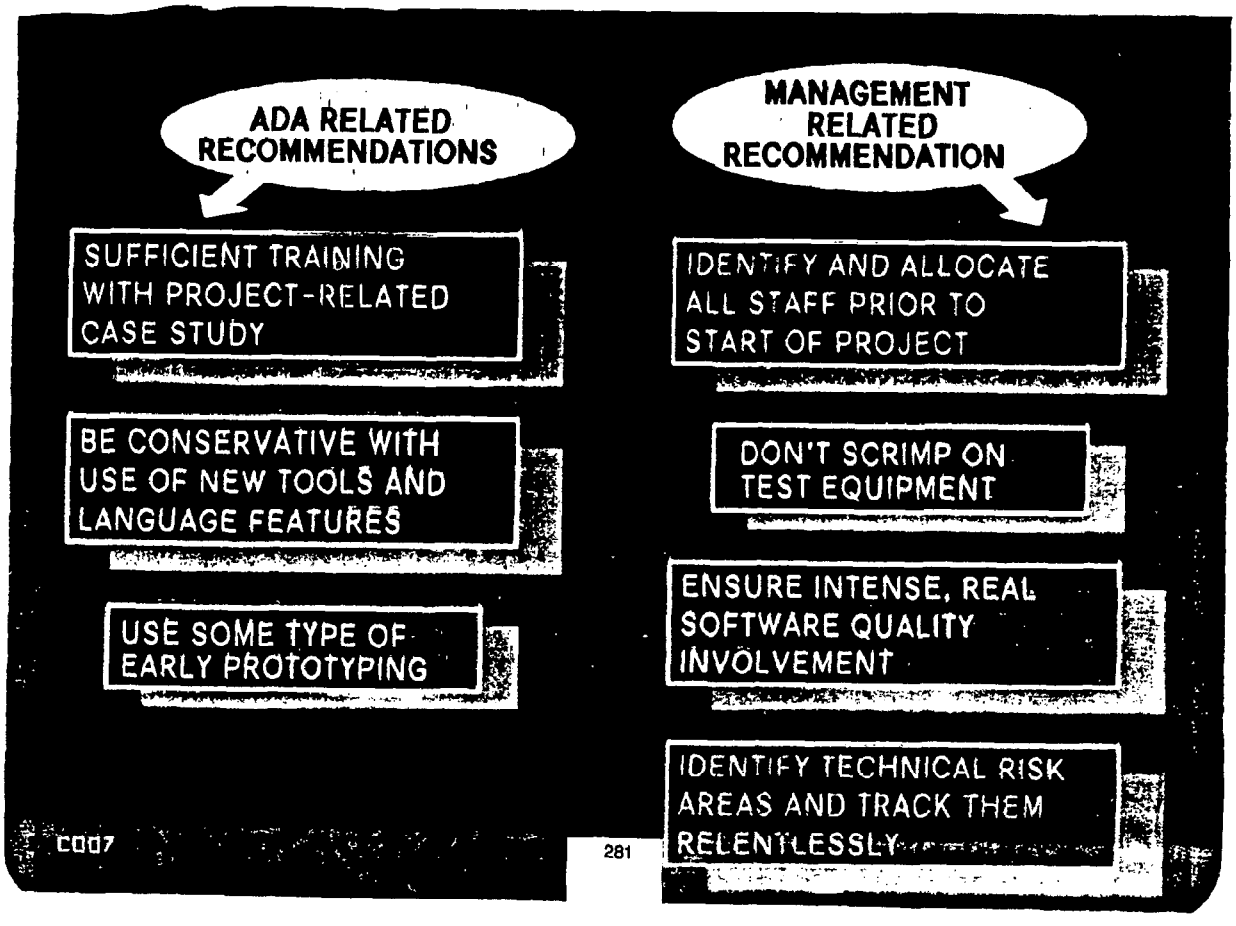
- -Select the BEST engineers available
- -Identify/allocate all engineers to be used
- -Select Software-literate Manager
- -Utilize Ada-specific software training
  - Intensive
  - Off-site
  - Meaningful case studies
  - All attend

C003


PLANNING
STAFFING/
TRAINING
EQUIPMENT
NEEDS
ADA LANGUAGE
USAGE
C004


PROCESSOR/COMPILER
SELECTION
RUNTIME
TECHNICAL
ISSUES
THROUGHPUT
UTILIZATION
NEW COMPILER
VERSION
C005


TOTAL SOFTWARE QUALITY
REVIEW DOCUMENTATION AND SOFTWARE
ADA TRAINING
TEAM MEMBER
C005


ADA RELATED RECOMMENDATIONS
SUFFICIENT TRAINING WITH PROJECT-RELATED CASE STUDY
BE CONSERVATIVE WITH USE OF NEW TOOLS AND LANGUAGE FEATURES
USE SOME TYPE OF EARLY PROTOTYPING
MANAGEMENT RELATED RECOMMENDATION
IDENTIFY AND ALLOCATE ALL STAFF PRIOR TO START OF PROJECT
DON'T SCRIMP ON TEST EQUIPMENT
ENSURE INTENSE, REAL SOFTWARE QUALITY INVOLVEMENT
IDENTIFY TECHNICAL RISK AREAS AND TRACK THEM RELENTLESSLY
C007