AUTOMATED TOOLS, ADA AND CUSTOMER REVIEWS: A CANDIDATE APPROACH

by Jerry H. Hendrix

Boeing Military Airplanes
Simulation and Training Systems
Huntsville, Alabama

Abstract

In today's fast-track technology explosion, the production of software for real-time systems is enhanced by the availability of technological advances such as automated tools and the Ada programming language. The use of these advances offers increased productivity and shorter schedules when used properly. The use of automated tools is helpful but may lead to a systems design that is decoupled from the software product. Some tools force users to use a methodology which leads to structures which may not directly translate to a higher order language software structure (for example, open ended if statements). Not using tools in tight-scheduled programs may lead to a paper-intensive development increasing schedules and cost. There are many tools available today that offer design and documentation assistance in every development phrase (Automated Requirements Analysis Tools, Graphical Object Oriented Design Tools, Boeing's own Automated Software Engineering). A proper Ada development methodology can offer assistance in every developmental phase. The dilemma facing software developers today is how to integrate the proper toolset and Ada approach to allow the best product and give the customer a full understanding of the development. The customer must assure correctness, completeness and consistency although he may not fully understand the target software and complexity of the Ada language. It is up to the contractor to expose the customer to the development in such a way as to allow him to properly critique the development activities. This paper will offer a candidate approach to coupling automated tools with Ada developments and customer reviews. It will present criteria for developmental tools selection, such as commonality of tools host to software development host, user interface considerations and methodology consideration. The paper will also present how these tools are integrated in a common environment, how an Ada development is enhanced by automated tools and how the customer can benefit from the proper mix.

There is a fundamental limit to the complexity with which a human can cope. Technically many problems can be solved whereas in reality the mismanagement of complex systems can cause software projects to fail. Many attempts have been made to solve the mismanagement of this complexity. New software and system analysis automated tools and the introduction of more English like languages like Ada are candidates to help alleviate this complexity. As the complexity of systems increase, so must the means by which this complexity is handled. "Darth Vader" like helmets for weapon aiming and voice recognition systems are examples of these complex systems. The user customer has to not only understand the complex systems but also the complex implementation of the simulator that follows. This paper refers to the user as those persons who are responsible, from a customers perspective for the technical correctness of the simulator. Those individuals at the training systems commands (ASD/YW, NTSC, PMTRADE and others) should be given an approach by the simulator vendor which exposes the implementation in the best possible way. This paper will explain how automated tools, and technological advances like the Ada lnaguage can assist in solving this complexity and will offer a candidate implementation approach coupling automated tools, the Ada language and the user customers reviews.

Automated Tools

Low cost hardware engines and superior software packages for automated tools offers new opportunities for designing and developing software. Until the present, software design and resulting documentation have been hand produced. Automated tools give the software developer an interactive pictorial solution to the problem space. Software developers relate easily to pictures and graphics. Pictures are used to present and explain all kinds of information. Charts and graphs become familiar and help assimilate and understand the data in business presentations, newspapers, and textbooks. George Raeder states that it is "commonly acknowledged that the human mind is highly visually oriented and that people acquire information at a significantly higher rate by discovering graphical relationships in complex pictures than by reading text".

Advantages of Graphical Representations

"Raeder describes several reasons why pictures have an advantage over text in providing information clearly and quickly.

Text is a sequential mode of expression. We must read or scan through preliminary information in order to find a particular description or explanation. Pictures offer us random access to data. Our eyes can rapidly move to any area of a drawing and locate required details. Important features can be highlighted to focus our attention quickly.

Text is one dimensional medium consisting of words. Pictures provide three dimensions for the description of information. Physical properties such as shape, color, texture, and size can be used to enrich the presentation. Because pictures can provide data with more variety, they can present the same data more concisely and compactly than text.

Pictures generally transfer data faster than text. The data can be accessed and understood quicker and with less effort than a textual description.

Pictures are often used to present abstract ideas in ways that make them simpler for people to comprehend. When trying to understand an abstract concept, many people come up with a mental image which enables them to understand the abstraction. A picture can present an already prepared image to a reader and speed up understanding of new or complex ideas"

Many varietes of graphical oriented approaches have been produced into automated tools. Some automated tools offer structured analysis approaches, others Object Oriented Design, others Buhrs' System Design Approaches, but which tool is right for the software developer.

The software developer must have a tool or set of tools which address total software development life cycle. The tool(s) must assist in the systems engineering process to the final system verification test. The tool(s) should:

- Trace requirements from System Specifications to lower level specifications
- Offer consistent verifiable approaches
- Assist document production
- Offer a means to a methodology and not a methodology

REQUIREMENT ALLOCATION

The requirement allocation and trace automated tool should provide the visibility to either trace a requirement from the system specification down to the individual system element that implements it, or to show for a system element how a particular requirement can be traced up the specification tree to the system specification. The tool should assume a specification tree that starts from the system specification and expandes regularly down to individual hardware and software end-item specifications, and, for software end-items, on down to top-level components within the software end-item.

Figure 1 illustrates a typical specification tree and specification decomposition. Note that each document has a single parent document, and that all documents flow ultimately from the system specification.

During system development, requirements in a specification may be <u>allocated</u> to a lower-level, as-yet-written subsystem or end-items specification. Likewise, when a specification is written, its requirements must be traceable either explicitly or as derived requirements from a higher level specification. Various automated schemes can be devised to simplify this bookkeeping (a large system specification will contain thousands of requirements, ultimately generating tens of thousands of lower-level requirements.) The goal of these schemes should be:

1. To show that all requirements at higher levels have been *successfully allocated to lower levels and ultimately to* individual hardware and software elements. In other words, the designed system meets all requirements.

2. To show that all requirements at a particular level do indeed tie back to a higher-level requirement

3. To assist the design review process by showing which lower-level specifications or end items implement system requirements, and how lower level requirements, particularly derived ones, relate back to the system specification.

4. To assist in assessing system changes, by being able to show for a system-level requirement what lower-level specifications and end-items are affected.

The major concern in the analysis and design phases is that all requirements are allocated to an end item and that allocation can be traced.

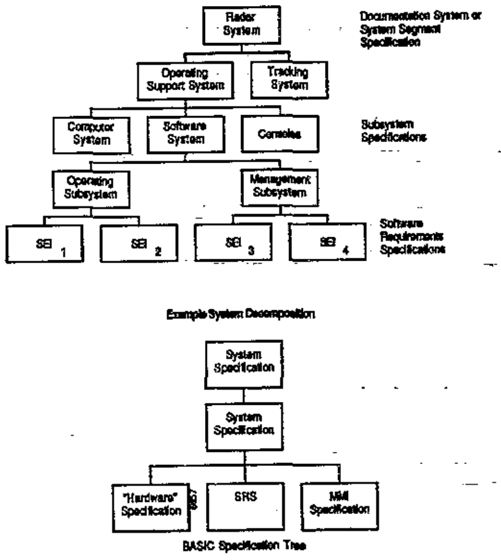


Figure 1
Specification Tree

VERIFIABLE APPROACHES AND DOCUMENTATION GENERATION

The tool(s) must offer consistent refinable approaches regardless of the development phases it represents. Tools that offer particular techniques must provide checking for consistency and completeness.

Assistance in the production of required documentation is needed to shorten the development process. The documentation should be extracted from portions of the automated analysis tools. The software developer should be transparent to document production. The only input needed by the software developer should be the boilerplate text for specifications and design documents. Boilerplate text is text that is used basically as is except for replacing unit specific keywords or phrases.

ARE TOOLS METHODOLOGIES?

The tool(s) should offer assistance to the software development approach but should not be the development methodology. Danger exists in tools which offer methodologies. These tools may not be tailored to the software design needs. Many tools will bind the implementation with structures which may not be consistent with design objectives, i.e., open ended if blocks for flow diagrams.

The use of these tools solve many laborious hours of building documentation and doing paper analysis but also introduce new problems. If many tools are integrated into a tool kit, the transition between tools and the data passing techniques must be 100% complete. The software contractor should strive to identify tools which can be hosted on a common environment. A single user entry point for automated requirements analysis tools and design tools will lead to an easier transition between requirements and design. If a tool kit is used in which these tools are hosted on different environments, the software developer may have many different terminal interfaces which are often times confusing causing non-productive.

Ada

Ada can also assist in alleviating these complex systems software development. The Department of Defense (DoD) directed the use of Ada on all weapon system acquisitions. Ada was chosen by the DoD to address the existing software crisis. The language improves software consistency, reliability and maintainability, improves productivity and reduces life cycle cost. The use of Ada allows enforcement of sound engineering principles. Those principles are: abstraction — extraction of essential details into an understandable unit; information hiding — details of an implementation or abstract made inaccessible; modularity and localization — grouping logically related items and completeness where no essential details are missing.

Ada was not designed to be just another programming language. It was designed to be a programming system that would have features to encourage modern programming practices. Ada offers features such as packages, types, data encapsulation, and generics that enable Ada to be a powerful tool to help in understanding problems and expressing solutions in a manner that directly reflects the multidimensional real world. Coding in Ada should be approached with the spirit it was designed in; with modern programming practices in mind. Ada offers flexibility in the implementation in that all development phases can be supported by the usage of the Ada language. Ada is so English-like that it can be used to express requirements, preliminary design, detailed design and code.

Figure 2 illustrates the reviews in a typically development. MIL STD 1521 series contains detailed information about reviews. This paper will only address the System Requirements Review (SRR), Systems Design Review (SDR), Software Specification Review (SSR), Preliminary Design Review (PDR) and Critical Design Review (CDR).

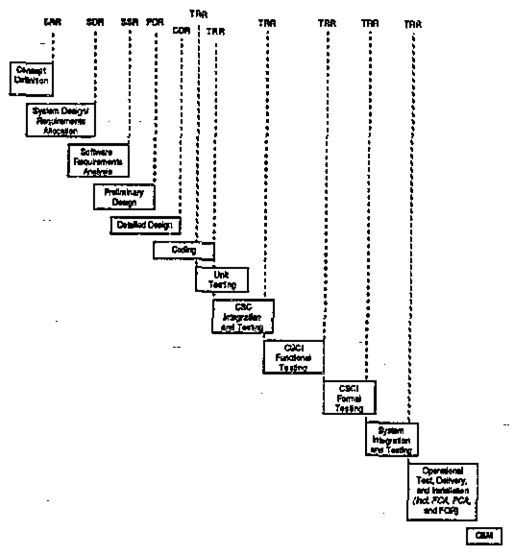

SRR
SDR
SSR
PDR
TRR
CDR
TRR
TRR
TRR
TRR
TRR
Concept Definition
System Design/ Requirements Allocation
Software Requirements Analysis
Preliminary Design
Detailed Design
Coding
Unit Testing
CSC Integration and Testing
CSCI Functional Testing
CSCI Formal Testing
System Integration and Testing
Operational Test, Delivery, and Installation (Incl. FCA, PCA, and FQR)
O&M

Figure 2
Developmental Reviews and Software Life Cycle

System Requirements Review (SRR)
The SRR shall evaluate the definition of system requirements, including those requirements planned for allocation to software.

a. The review shall include the following:
   - System concept
   - Requirements to be satisfied by software
   - Requirements to be satisfied by hardware
   - Rationale for the concepts presented

b. Data to be reviewed shall include:
   - System study reports
   - Trade study reports
   - System specification

System Design Review (SDR)
The SDR shall evaluate the allocated system requirements for completeness, traceability, optimization, and risk assessment.

a. The review shall include a presentation of the initial allocated software functional performance requirements and software verification requirements.

b. Each software requirement shall be reviewed explicitly.

c. Data which support the software allocations in relation to the total system requirements should be presented for each software requirement.

d. All software external interface requirements shall be reviewed.

e. Data to be reviewed shall include:

- Segement Specification
- Man-machine interface specification
- Trade study reports - summary of software recommended requirements to satisfy the accepted system specification.
- Draft software requirements specification - derived from study and supported with rationale.
- Draft interface requirements specification - summarize the interface requirements and potential problems.

Software Specification Review (SSR)

The SSR shall evaluate the software requirements, as specified in the *Software Requirements Specifications (SRS)*, Interface Requirements Specifications (IRS).

a. The review shall be devoted to reviewing all software requirements.

b. All other organizations that impose requirements on the software should participate.

c. The following shall be reviewed for each functional requirements allocated to software.

- Traceability of the requirement or derived requirement from the system specification through subordinate specifications.
- Rationale for derivation of derived requirements.
- Completeness of the implementation algorithms and equations.
- Testability of the requirements as stated.
- Availability of constants and tables required for calculations.
- Realism of computation accuracy requirements.
- Consistency in the use of symbols and equations.
- Compatibility with external interfaces.

d. IRSs when applicable should be reviewed for completeness. The level of detail should vary depending on whether or not the CSCI interface is to an existing component. Information from the IRSE should allow:

- Name
- Type (digital, discrete, analog)
- Range
- Units
- Accuracy
- Frequency (data transfer rates)
- Error checking requirements
- Byte number
- Data type
- Size
- Scaling/conventions

e. Deficiencies in review items shall be assessed for impact on development and test schedules.

f. The review shall not be considered complete until a plan and schedule for correcting any deficiencies has been established.

Preliminary Design Review (PDR)

The preliminary design review provides a positive demonstration that the selected design approach will result in a CSCI that meets the requirements set forth in the SRS. Most PDR review material will have been produced by the preliminary deisgn activity, such as: software functional flow, storage allocation charts, timing and sequencing, operating modes, program hierarchical structure, data base design and man-machine interfaces. Designers are usually required to supplement this material with viewfoils summarizing significant design features and to be prepared to answer questions brought to the meeting by the reviewers.

Critical Design Review (CDR)

The critical design review provides a positive demonstration that the CSCI logic design will satisfy the requirements set forth in the SRS that coding may proceed.

Most CDR review material will have been produced by the detailed design activity, such as: TLCSC interface block diagrams, Program Design Lanugage (PDL) logic flows, time lines, storage and timing allocation charts and data base description. The detail is such as to present the detailed design.

Once a proper understanding of the advantages of automated tools, Ada and customer reviews are obtained, a candidate model approach can be hypothesized. In the correct integration of these complexity minimizing contributiors the customer user gains insight into the entire developmental process.

The following will present a candidate that shows how integrating automated tools, Ada and the customer reviews will decrease schedule, cost, and documentation and increase understanding by the procurers of the simulator.

Boeing uses a set of vendor supplied automated tools called Boeing Automated Software Engineering (BASE). BASE provides a tool kit which does not force a methodology but allows methodologies to be tailored to specific applications. These set of tools are integrated by Boeing.

The automated tools are provided to the developer by the BASE project's Software Integrated Environment (SwIE).

The need for software development tools is widely recognized by aerospace firms. They realize that a methodology promotes software engineering productivity improvements, improves software quality, and decreases the uncertainties associated with the software development schedules and overall software reliability. Therefore, there is a thrust to arrive at methodologies, and tools which support those methodologies. The SwIE is the effort within the Boeing Company to provide tools.

The SwIE provides an integrated software development environment which supports the tasks specified by DOD-STD-2167 and 2167a and automates the production of software products. Standardization on the software lifecycle framework, tasks, and product standards enables development of a common set of tools for all software projects.

General Description

The BASE Software Integrated Environment (SwIE) supports the life cycle phases as defined in DOD-STD-2167 and 2167a as illustrated previously. The capabilities of the BASE SwIE include:

a. REQUIREMENTS. Requirements are evaluated by diagnostic analysis. A prototype is built using CADRE, Inc. teamwork tools. Requirements are allocated and tracked using Boeing's Requirements Tracking and allocation tool (RT2).

b. DESIGN. The software architecture, interfaces, data base, and programs are designed with Teamwork. The designs are evaluated from diagnostics from the design tools and modeling tools.

c. CONSTRUCTION. The generation, integration, and analysis of code and modules is supported by editors, compilers, linkers, and analyzers.

d. TEST. The testing of code is supported with downloaders from the ASSIST package.

e. GENERAL SUPPORT. The integrated generation of documents and graphics is supported by the documentation tools DOC and PicED. The user can plan and monitor the process and resources (e.g., estimate schedules, budgets, productivity, quality, and risk analysis) using the management and task planning tools.

f. CONFIGURATION MANAGEMENT. The configuration managers manage product configuration of code, libraries, and documents, and track and report changes.

Figure 3 illustrates how the SwIE capabilities apply specifically to the software development phases. Object orientation is one of the hottest methodologies in today's Ada world. The object orientation organizes data into manageable software pieces. Ada, as a software engineering tool and object orientation, promises to increase programmer productivity, product reliability, and software reusability. The software development approach consists of two distinct but integral parts, the methodology for software development and the structural model for software consistency.

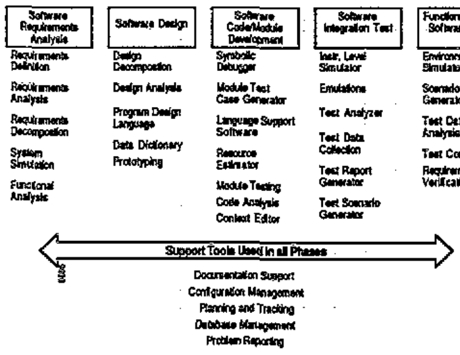


Figure 3
Overview of BASE Support

The methodology must fit the problem domain. There are many methodologies, some are consistent and some are inconsistent with the production of real-time efficient software. The methodology must manage the complexity of the proboem. Application of Ada to a "textbook" type problem is trivial, but applying Ada to a large complex system requires a different mindset. The methodology must encompass and couple the phases of the software lifecycle.

If the method addresses each development, it can increase productivity and product reliability as well as removing the need for unwanted documentation. The method must also provide uniformity into the software design and provide enhanced communication. In order to gain maintainability as a design, uniformity of the design enhances understandability of the system as a whole and its associated objects. The method that enforces communication via procedure or function calls alleviates the problem associated with change. The data in and out of an object is readily available and controlled. The enforced communication provides ease in maintenance. The understandability of an abstraction is increased when there are not unwanted interfaces. Choosing a method consistent with real-time development is also important, although the method must provide reusable, reliable, and maintainable software. The methodology in itself is not enough, although an essential part!

The structural model enforces a consistency in the software structure, thus enforcing an understanding. Ada is a robust language and misuse of the language can result in fragmented code such as standalone procedures, functions, and disconnected mini-data pool data packages. The production of the structural model is attributable to an understanding of the requirements, understanding Ada, and the scope of tradeoffs (number of processors memory, etc.). The structural model becomes the architecture for the designers. In essence, the structural model consists of package structure, contents, and interfaces.

DEVELOPMENT PHASES IN ADA

The classical software development phases as described are: requirements, design, code, test, integration and acceptance.

REQUIREMENTS

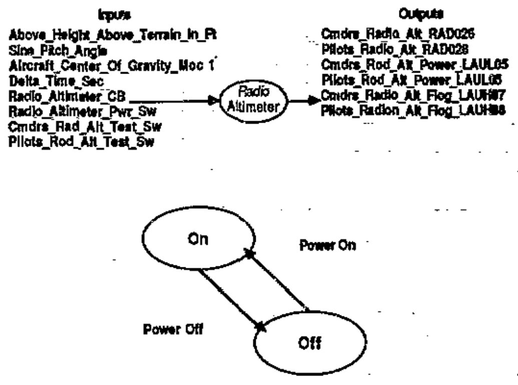
At System Requirements Review time, a consensus is supposed to be reached on the definition of requirements for the program. In the past, these reviews have defined specific requirements for the program. In the past, these reviews have defined specific requirements which may or may not be fully met by the contractor. Requirement Traceability Matrices (RTM) are set up to ensure the transition or requirements to design. Ada now offers a new means of dealing with requirements analysis and allocation. Figure 4 shows an example of reqiurements analysis in Ada. Notice that the requirements correlate directly into Ada code, and thus compilable and consistent. There are no absolutes, even with Ada. An understanding of the requirements still must be discussed and unknown or vague requirements explicitly understood. There is still a place for the RTM in an Ada design. The definition of requirements in a single or multiple Ada package makes those requirements consistent and thus affords the allocation of those requirements using the data flow analysis to identify or allocate subprograms or packages.

```
type    STEADY_STATE_WIND_CONTROL  IS
        (STANDARD_ATMOSPHERE_WINDS_ARE_SELECTED,
        STEADY_STATE_WIND_CHANGES_ARE_SELECTED
        STEADY_STATE_WIND_LAPSE_RATE_CHANGES_ARE_SELECTED,
        FAA_WIND_PROFILES_ARE_SELECTED,
        GLOBAL_WIND_NORMAL),

type    FAA_APPROVED_WIND_SHEAR_PROFILES IS
              (NEUTRAL_LOGARITHMIC,
              FRONTAL_1_TOKYO_1966,
              THINDERSTORM_FAA_MATHEMATICAL,
              FRONTAL_2_LOGAN,
              THUNDERSTORM_2_PHILADELPHIA,
              THUNDERSTORM_3,
              THUNDERSTORM_4,
              THUNDERSTORM_5,
              FRONTAL_3,
              THUNDERSTORM_6_JFK),

subtype  FAA_WIND_SHEAR_PROFILES_HEAD_AND_CROSSWINDS IS
        FAA_APPROVED_WIND_SHEAR_PROFILES range NEUTRAL_LOGARITHMIC, .
        FRONTAL_2_LOGAN

subtype  FAA_WIND_SHEAR_PROFILES_HEAD_CROSSWIND_AND_VERTICAL IS
        FAA_APPROVED_WIND_SHEAR_PROFILES range
        THUNDERSTORM_2_PHILADELPHIA, . THUNDERSTORM_6_JFK:
```

Figure 4
Requirements in Ada

DESIGN AND CODE

These two phases represent the most dramatic shift in the software development phase while using the Boeing methodology. The software transition is no longer from design-to-code, but from requirements-to-design/code. In the methodology, the design is laid out in Ada PDL and compiled at each tier level. The design is code and the code is compilable, thus making the design consistent. In Ada, the interfaces can be enforced if the methodology allows. The interfaces are defined in Ada code and compiles the design, thus making the interfaces consistent. The interfaces are identified by using data flow analysis for interface definition. The design continues by using control flow analysis to identify the state in which a system may reside. Figures 5 gives an example of a design in Ada using the results generated by the automated tools.



```
begin
  Radio_Altimeter_Power : = CB_State
     (Radio_Altimeter_CB)=Energized and Switch_Status
     (Radio_Altimeter_Switch)=Latched;
  If Radio_Altimeter_Power then
          radio_altimeter_power_on ;
  else
          radio_altimeter_power_off ;
  end if;
end radio_altimeter;
```

Figure 5
Ada Design From Automated Tools

SOFTWARE INTEGRATION AND TEST

There is not a classical phase called software integration (SWI) in Ada. *SWI is nebulous in Ada because the SWI is* handled at compilation. When something is compiled in Ada, it is integrated with that system. Once a tier has been designed/coded, testing of that unit is started. A two-segment testing approach is used. White Box Test and Black Box Test. White Box testing is used to check outputs from a unit while stimulating that unit with known inputs. A unit is the lowest level of abstraction in this application. After successful White Box testing, the unit is integrated with the rest of the system and Black Box testing begins. Black Box testing is stimulating a unit while it is integrated by looking at inputs and how that unit responds to thos inputs. Comparisons are then made between Black Box and Whit Box test results.

The Government will gain more visibility into the software development process than in the past. The Government will have the capability to understand the design process and the software produced from this approach. The methodology lends itself to the best solution and the structural model forces consistency in the Ada software product. The production of software in the past has focused on the transition between an activity called design to code. In preparation for PDR and CDR, many hours are spent drawing block diagrams or writing pseudo-code to explain a full understanding of the design. Once that design is approved, coders come in and code the software structure and the design at PDR and CDR has been lost. There was communication, but there was not an understanding between designers and coders. In today's Ada world, the software transition is from requirements analysis to design which is compilable Ada code. Now there is communication and understanding. The software design can be separated from the physics of a problem, allowing compilable software detailed design documentation (SDDD) to be presented at CDR in Ada PDL. Upon Government design approval, the functional *unit is the only coding job left. Interfacing and localiza*tion of errors is seen and enforced when presented at CDR. The SDDD can be viewed as enforceable documentation with specification to the unit level or a "code to" unit requirement. The job left for coding is to insert the math models into the design. Figures 6 thru 10 illustrate the concept, coupled with automated tools for the requirements analysis design and code phases.

This paper has attempted to present a candidate approach coupling automated tools and Ada with customer reviews. Not using tools in tight-scheduled programs may lead to a paper-intensive development increasing schedules and cost. There are many tools available today that offer design and documentation assistance in every development phase (Automated Requirements Analysis Tools, Graphical Object Oriented Design Tools, Boeing's own Automated Systems Engineering). A proper Ada development methodology can offer assistance in every developmental phase. The dilemma facing software developers today is how to integrate the proper toolset and Ada approach to allow the best product and give the customer a full understanding of the development.

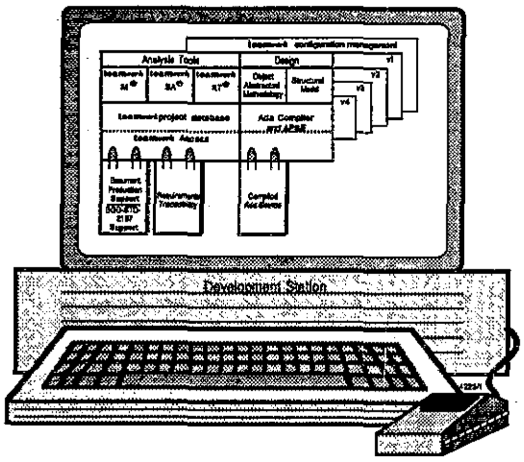
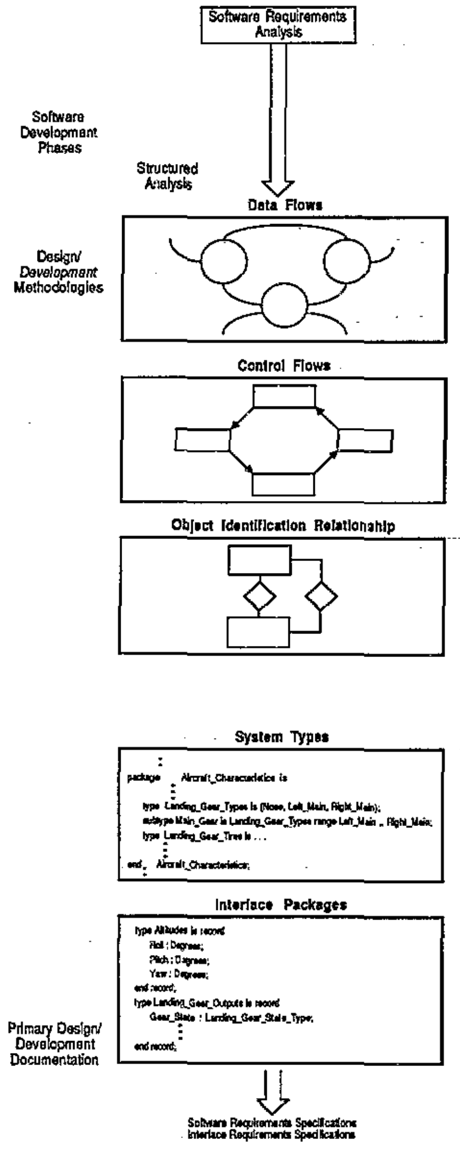

teamwork configuration management
Analysis Tools
Design
teamwork IM
teamwork SA
teamwork RT
Object Abstracted Methodology
Structural Model
v1
v2
v3
v4
teamwork project database
Ada Compiler
teamwork Access
Document Production Support DOD-STD-2167 Support
Requirements Traceability
Compiled Ada Source
Development Station

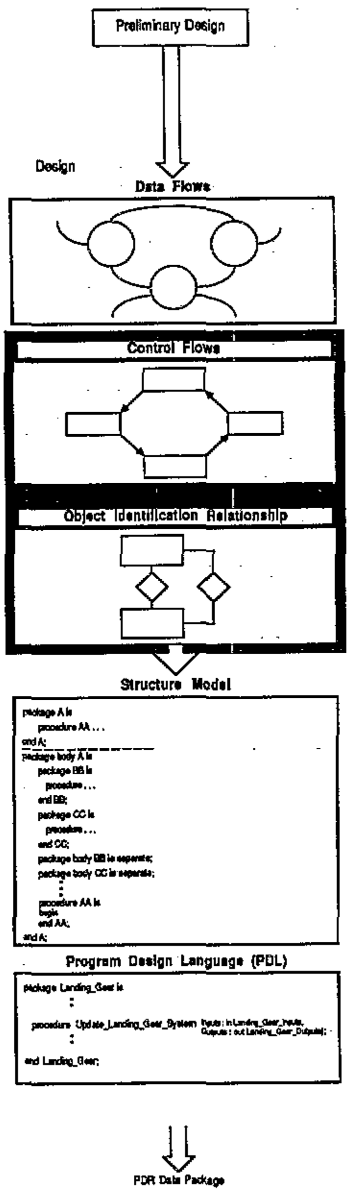
Figure 6
Integration and Development on Common Station


Software Requirements Analysis
Software Development Phases
Structured Analysis
Data Flows
Design/ Development Methodologies
Control Flows
Object Identification Relationship
System Types
Interface Packages
Primary Design/ Development Documentation
Software Requirements Specifications
Interface Requirements Specifications

```
        :
package    Aircraft_Characteristics is
          :
   type  Landing_Gear_Types is (Nose, Left_Main, Right_Main);
   subtype Main_Gear is Landing_Gear_Types range Left_Main .. Right_Main;
   type  Landing_Gear_Tires is . . .
         :
end    Aircraft_Characteristics;
```

```
type Attitudes is record
   Roll : Degrees;
   Pitch : Degrees;
   Yaw : Degrees;
end record;
type Landing_Gear_Outputs is record
   Gear_State : Landing_Gear_State_Type;
          :
end record;
```

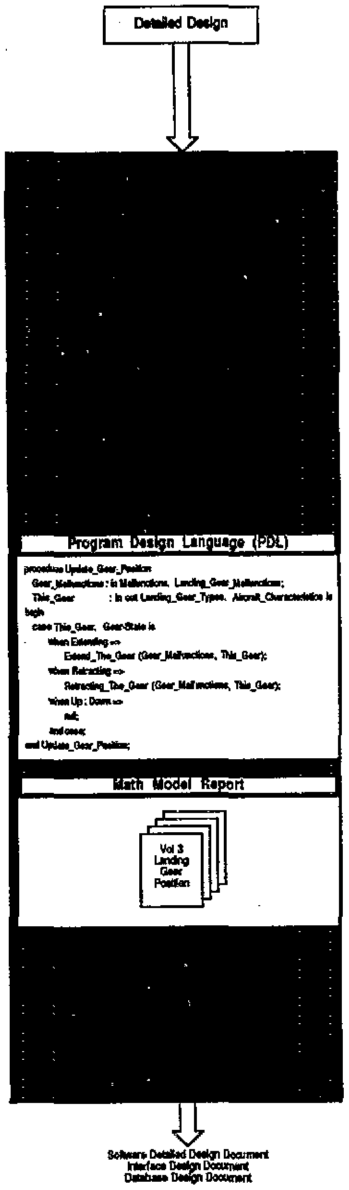
Figure 7
Design and Development Methodology



```
package A is
    procedure AA . . .
end A;
package body A is
    package BB is
      procedure . . .
    end BB;
    package CC is
      procedure . . .
    end CC;
    package body BB is separate;
    package body CC is separate;
          :
    procedure AA is
    begin
    end AA;
end A;
```

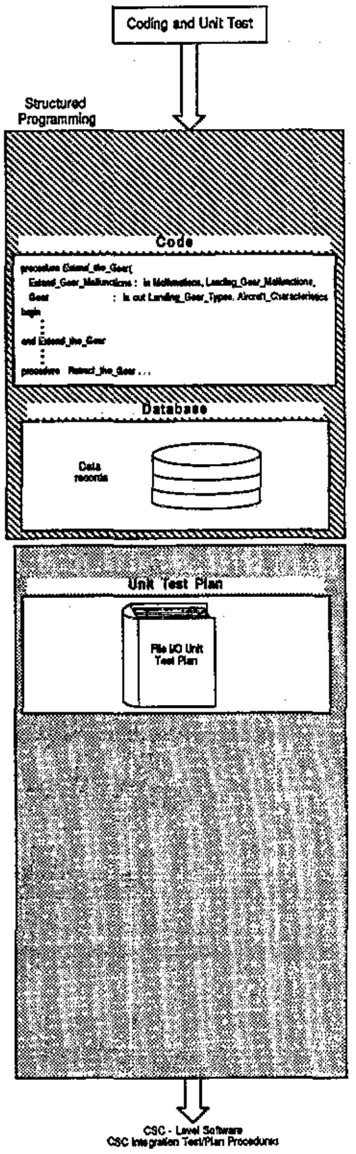
```
package Landing_Gear is
          :
  procedure  Update_Landing_Gear_System  Inputs : in Landing_Gear_Inputs,
                                         Outputs : out Landing_Gear_Outputs);
          :
end Landing_Gear;
```

Figure 8
Design and Development Methodology


Detailed Design
Program Design Language (PDL)
Math Model Report
Vol 3 Landing Gear Position
Software Detailed Design Document
Interface Design Document
Database Design Document

```
procedure Update_Gear_Position
  Gear_Malfunctions : in Malfunctions, Landing_Gear_Malfunctions;
  This_Gear         : in out Landing_Gear_Types, Aircraft_Characteristics is
begin
  case This_Gear, Gear-State is
      when Extending =>
          Extend_The_Gear (Gear_Malfunctions, This_Gear);
      when Retracting =>
          Retracting_The_Gear (Gear_Malfunctions, This_Gear);
      when Up : Down =>
          null;
      end case;
end Update_Gear_Position;
```

Figure 9
Design and Development Methodology



```
procedure Extend_the_Gear(
  Extend_Gear_Malfunctions :  in Malfunctions, Landing_Gear_Malfunctions,
  Gear                     :  in out Landing_Gear_Types, Aircraft_Characteristics
begin
    :
end Extend_the_Gear
    :
procedure   Retract_the_Gear . . .
```

Figure 10
Design and Development Methodology

REFERENCES

1. The Boeing Company "Boeing Automated Software Engineering (BASE) Software Integrated Environment (SWIE) Software Users Manual," June 9, 1986, p17-18.

2. The Boeing Company "Boeing Embedded Software Standards," January 19, 1987

3. The Boeing Company "Boeing Automated Software Engineering (BASE) RT2 Tutorial, December 31, 1987, p7.

4. Hendrix, Jerry H. "The Next Generation of Trainers: Lessons Learned from the Ada Simulator Validation Program.," Preceedings from the 9th Interservice/ Industry Training Systems Conference, November 30 - December 1987, Washington, D.C., p62-66.

5. Myren, Greco "Experience in Implementing on Ada Real-Time Program for Flight Simulation Operation" Preceedings from the 9th Interservice Industry Training Systems Conference, November 30 - December 2, 1987, Washington, D.C., p68.

6. Parrish, William F. and Woodard, Pamela S. "Feasibility of a Graphical Design for an Ada Software Development," Preceedings from the 10th Interservice Industry Training Systems Conference, November 29 - December 1, 1988, Orlando, FL, p35-36.

ABOUT THE AUTHOR

Jerry H. Hendrix is a systems software engineer with Boeing Military Airplanes in the Simulation and Training Systems organization. He has been responsible for systems and software work on several Boeing projects including the Ada Simulator Validation Program. He is currently involved in the Fiber Optic Guided Missile Program and research and development activities on Ada real time systems/software development. Mr. Hendrix has been involved in Ada development for five years and holds a Bachelor of Aerospace Engineering Degree from the Georgia Institute of Technology.