Ada and Hatley-Pirbhai

Dean Roberts
Hughes Santa Barbara Research Center
d.roberts@ieee.org

Rod Ontjes
Hughes Santa Barbara Research Center

Abstract-- This paper will introduce NASA STD 2100, a tailorable software documentation standard. This standard is tailored by altering the structure of the Software Documentation Set. This set is comprised of four parts: Management Plan; Product Specification; Assurance and Test Procedures; Management, Engineering, and Assurance Reports. There is information associated with each one of these sections.

Each one of these sections consists of two parts: Basic Information and Data Item Description (DID) specific information. Basic Information is the same for all DIDs and consists of subjects like the Introduction, Related Documents, or Glossary.

The amount of information present in the DID specific information portion tailors the standard for specific projects. Chapters in the DID specific information section can remain as chapters, as is the case for small projects, or they can be "rolled-out" into their own separate volume for larger projects.

Next, the Hatley-Pirbhai method for Structured Systems Architecture is introduced as a technology and language independent method for developing software.

Requirements analysis is performed by means of data flow diagrams (DFD) and a data dictionary to define the data flows. These DFDs can have information notes attached to them to facilitate understanding of the design.

The DFDs are broken down into Process Specifications (PSPECs) which define the data transformations that take place within the primitive processes.

Once the requirements analysis is performed, an architecture model is created by allocating processes to architecture modules. Because these processes contain the requirements, traceability from requirements to design is maintained.

The architecture modules are broken down into a structure chart. The "leaves" of the structure chart a described by a module specification (MSPEC). The MSPEC can be text, graphics, or pseudo-code. The MSPEC also forms the foundation for Unit Testing.

When used with Ada, the units specified by the structure chart do not necessarily have a one-to-one correspondence with Ada packages. It is preferable to illustrate design highlights with the structure chart than to have an obscure design but with a nice mapping to Ada.

A breakdown from context diagram to lowest DFD is given.

Following the introduction of the documentation method and the design methodology, an example which employed both of these tools is given. The example given is taken from the Flight Software of the Tropical Rainfall Measuring Mission's Visible and Infrared Scanner.

The last item shown is how Ada is mapped onto the detailed design portion of the Hatley-Pirbhai structured systems architecture methodology. Most commonly, the Hatley-Pirbhai structured systems architecture methodology is used for the design up to the architecture level but Ada specific tools are used for the detailed design.

The approach taken in this paper differed from that by using a strictly Hatley-Pirbhai approach.

In summation, this paper will introduce a NASA software documentation standard which can be tailored to many differently sized projects. It will also introduce the Hatley-Pirbhai design methodology. This is followed by a real-world example of how the documentation standard and the design methodology are used.

©1994 ACM 0-89791-684-0/94/0006 3.50

INTRODUCTION

The Hatley-Pirbhai Structured Systems Requirements and Structured Systems Architecture[4] methods have found use within the aerospace community as a way of developing complex systems in a disciplined manner. Many of the systems designed using this methodology are software systems, and quite often, those software systems employ Ada as the language of implementation.

Additionally, it is increasingly common to find a technical documentation standard specified for projects performed for the National Aeronautics and Space Administration (NASA) or the Department of Defense (DoD). In the case of the DoD, the well-known MIL-STD-2167A is commonly applied. NASA has used several types of standards but recently NASA-STD-2100[9] is being specified as the standard of choice.

This paper is concerned with the implementation of a Hatley-Pirbhai based design using Ada as the language and NASA-STD-2100-91 as the documentation standard. The intent is to show an example of how the Hatley-Pirbhai design methodology can be used with Ada.

The first section of the paper is concerned with introducing NASA-STD-2100. This standard has only had a formal reference document available since 1991, so it is quite possible that many engineers who might need to use it have not heard much about its details. The introduction will give a brief overview of its main attributes (scalability, application to software only, differences from MIL-STD-

2167) and how it can be used on many different types of software projects.

The next section talks about the Hatley-Pirbhai method briefly, and mostly about implementation issues. This section is provided to give the reader background into the philosophy of implementing a structured systems architecture without making this paper a tutorial on the Hatley-Pirbhai method. This section includes a discussion of what constitutes a "unit".

An example follows the Hatley-Pirbhai discussion. This example is based upon the flight software written for the Tropical Rainfall Measuring Mission's Visible and Infrared Scanner. This example goes from the initial requirements analysis to the module specification. It also discusses how the various design sections were documented under the NASA-2100 standard. Following this is a section that discusses how Ada was mapped onto the Hatley-Pirbhai design methodology. Mapping Ada onto Hatley-Pirbhai is no small task if one wants to stay within the Hatley-Pirbhai Structured Systems Architecture approach. The most common solution is to use the Structured Systems Architecture approach to generate the design up to the architecture level and then use an Ada-oriented approach (i.e. Booch diagrams) to design the structure of the code. The approach taken in this paper is to use the Hatley-Pirbhai approach for the entire design.

The conclusion sums up the paper by stating what the salient features of NASA-STD-2100 are and how these features can be used on projects of different sizes. It also states some general rules that can be used when using the Hatley-Pirbhai design methodology and Ada.

THE NASA 2100 SOFTWARE DOCUMENTATION STANDARD

The NASA software documentation standard NASA-STD-2100-91[9], called NASA-STD-2100 for short, can be applied to all NASA software. NASA-STD-2100 is only a documentation standard levied upon software. As such, there are no assumptions made as to the engineering processes. Indeed, there is no assumption of an engineering process being used at all. This differs from the DoD 2167A standard[5] which specifies both a development and documentation process.

The major driver behind the creation of NASA-STD-2100 is the desire to have all of the software documentation that is used by NASA to have the same basic framework. This allows NASA to easily judge the completeness and delivery of the documentation.

NASA-STD-2100 has, as one of its primary attributes, the ability to be scaled to match the size of the project for which software is being developed. This can greatly reduce the cost of implementing a software project. The documentation can be scaled by changing its most basic unit, the Data Item Description(DID).

The structure of a DID is shown along with a sample DID in Figure 1. All DIDs contain certain basic information which is indicated by the box pointed to by the arrow in Figure 1 labeled "Basic Information". This information allows the user to place the DID into the hierarchy of the other documents. It also provides information to the user explaining terms used and assumptions made inside of the DID. Each one of the titled sections in the figure occupies a single chapter. The DID specific information varies from DID to DID. Each section in the DID specific section also occupies a separate chapter.

The sample DID given in Figure 1 is the DID used for the Software Documentation Set. This is a top level view of all of the software documentation associated with a given NASA project. This set consists of four parts: Management Plan; Product Specification; Assurance and Test Procedures; Management, Engineering, and Assurance Reports. In order for the software documentation to be complete there must be some information associated with all of the different sections.

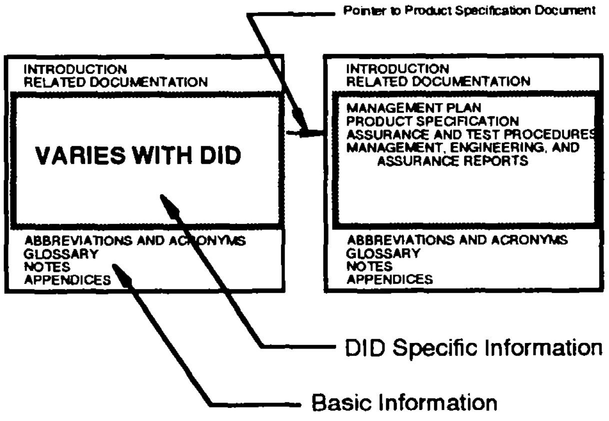

Figure 1. DID Structure and Sample DID.

The information placed in these sections can be one of four things: N/A, TBD, the complete information, or a pointer to another document. If the section does not apply to the document, then "not applicable" (or N/A) should be entered under the appropriate heading along with a reason as to why this information is not applicable. If the section has not been decided, then a "To be Decided" (or TBD) should be entered under the chapter heading. If the information is available and is small enough to be included in this particular document, then the information should occupy the chapter. If there is too much information to be included in the document, then a pointer should be placed under the chapter heading. This pointer refers to a document that is not written to NASA-STD-2100 or to a chapter that has become a stand-alone document.

A chapter can become a stand-alone document by wrapping the DID's Basic Information around the chapter. This process is called rolling out a document . For example, Figure 2 shows the process of rolling out the Product Specification into another document. To do this, the information contained in the Product Specification Chapter in the original DID is taken and the Basic DID information is wrapped around it. The Software Documentation Set has a reference to this new document under the Product Specification chapter.

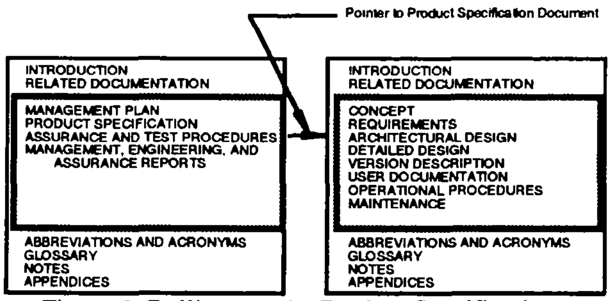

Figure 2. Rolling out the Product Specification.

This process could continue for any one of the DID specific topics in the now rolled out Product Specification document. For example, a separate Requirements document could be rolled out of the Product Specification document which was itself rolled out of the Software Document Set document. This allows NASA-STD-2100 to be tailored to many different sized projects by changing the number of documents rolled out of the original Software Documentation Set. The one thing that all of these documents have in common is the Basic Information Sections.

Basic Information Sections: Introduction- The Introduction section of the DID identifies the document, its scope, objective, status, and schedule. Additionally, the document's organization is also detailed. This organization, usually represented by a picture, denotes both what document you are looking at as well as how the document fits into the overall document hierarchy. The document organization for the Tropical Rainfall Measuring Mission Visible and Infrared Scanner's Flight Software[7] is shown in Figure 3. This figure was taken from the flight software's Product Specification document.

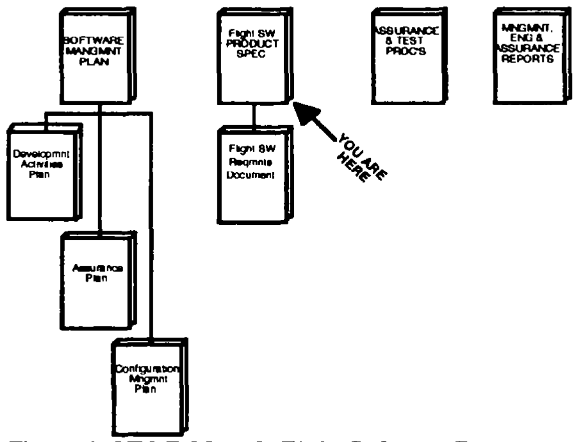
SOFTWARE MANGMNT PLAN
Flight SW PRODUCT SPEC
ASSURANCE & TEST PROC'S
MNGMNT, ENG & ASSURANCE REPORTS
Developmnt Activities Plan
Flight SW Reqmnts Document
YOU ARE HERE
Assurance Plan
Configuration Mngmnt Plan

Figure 3. TRMM VIRS Flight Software Document Organization.

The top row of four figures in the upper row indicate the documents needed for the Flight Software's Software Documentation Set. Figure 3 shows that each of the component documents in the Software Documentation Set has been rolled out into a separate document. Additionally, the Requirements document has been rolled out of the Product Specification (arrow in figure) which was itself rolled out of the Software Documentation Set. This figure succinctly places the document's position with respect to the other documents in the set as well as giving the user a good idea of what other documents will be required to complete the documentation set.

Basic Information Sections: Related Documentation- This section of the DID identifies the documents that are either referred to in the document or provide background information. The three types of documents listed in this section are: Parent Documents, Applicable Documents, and Information Documents.

Parent Documents can be either the documents that the current document was rolled out of or they can be external documents (performance specifications). Using the example in Figure 3, the Product Specification would reference the Software Management Plan and the Software Documentation Set. These would be examples of citing documents that are related to the source of the Product Specification. In addition to this the performance specification and the Interface Control Document were the external documents cited as parent documents.

Applicable documents are those which are referenced inside of the document. Examples would be military and commercial standards, user's manuals, or policy statements. Any reference made inside the document must have a corresponding parent or applicable document. Typically, contractors are required to supply NASA with copies of any documents considered parent or applicable documents. The exceptions to this, of course, are documents which come from NASA.

Information documents merely provide background helpful in understanding the contents of the document. There are no direct references made to the information contained within the Information documents, but it is assumed that the reader has knowledge of the information contained within those documents.

Basic Information Sections: Abbreviations and Acronyms- This section contains an alphabetized list of definitions, acronyms, and abbreviations used in the document. One common approach to creating a list of abbreviations and acronyms is to have a list inside of the Software Documentation Set and then provide pointers in the rolled out documents to this list.

Basic Information Sections: Glossary- This section contains an alphabetized list of specialized definitions and terms used in the document. The glossary list can also be treated in a manner similar to the abbreviations and acronyms section.

Basic Information Sections: Notes- Any information that is not contractually binding but aids in understanding can be placed here. Examples would be a memory map for a processor, or directions on how to use a crucial piece of equipment.

Basic Information Sections: Appendices- Information that is too large, detailed, or sensitive to be placed in the main text should be placed here. Examples would be schematics, panel layouts, or algorithm proofs. These materials can be bound separately but they are considered a part of the document for configuration purposes.

DID Specific Information- The information contained in this section is considered the "meat" of the document. The information required differs markedly depending upon the DID and the number of documents rolled out. It is beyond the scope of this introduction to detail the specific information contained in every section. That detail can be found within the standard itself[9]. Information for beginners on tailoring the documentation standard can be found in [3]. Details on the construction and maintenance of a software management plan can be found in [6].

HATLEY-PIRBHAI DESIGN METHODOLOGY

The Hatley-Pirbhai method for Structured Systems Design, with commercial computer aided system/software engineering (CASE) tools, support the engineering process throughout the system and software life cycle. The method is a communications tool, a management tool, and when coupled with a disciplined engineering process, produces excellent design documentation. From requirements analysis to design, integration and test, the method supports NASA and DoD standard documentation requirements and produces a coherent software product that addresses testability, requirements traceability and long term maintenance. This engineering process is useful for any application regardless of technology or language used.

Requirements analysis utilizes data flow diagrams (DFD) and a data dictionary, implemented according to the Hatley-Pirbhai method, to build a technology independent model of the system or software requirements. Employing data hiding, coupling and cohesion, an efficient model of the requirements is produced. External to the Hatley-Pirbhai method, conventions referred to as Information Notes are employed which are attached to each DFD (this is a feature of the CASE tool). The Information Note contains explanatory text that helps the reader (customer reviewer, designer, or maintenance personnel) to understand the diagram. Process Specifications (PSPEC) are utilized in the method to define each primitive process in the DFD. The PSPEC defines the data transformation that occurs within that process. Requirements statements included with the PSPEC succinctly define the software requirements. While requirement statements are not a part of the Hatley-Pirbhai method, when included in the model they aid the generation of requirement specifications that are comprehensive and illustrative yet meet the documentation standard requirements for individually identified requirements.

Each requirement statement must be a short, unambiguous statement that defines a requirement. The statement must be testable. Each statement is uniquely numbered for ease of identification. Within the CASE tool, conventions have been developed for documenting requirement statements and the traceability information for that requirement that allow automated generation of traceability matrices.

This procedure for requirements analysis provides a disciplined environment for interface requirements definition, system or software requirements analysis and provides the capability for automated requirements specification generation. These specifications include the DFDs, with explanatory text, the requirements statements and the data dictionary.

The engineering performed and the documentation produced provides the foundation for the acceptance test procedure where the requirements will be validated. The requirements also provide the foundation for the architecture design effort which is the next step in the life cycle. The key elements of testability and requirements traceability are an integral part of this initial step allowing them to be addressed very early in the process.

During architecture design, the transition from a technology independent model to a technology dependent model begins. Design rules, experience and heritage designs guide the allocation of requirements model processes into architecture modules. The allocation of processes to architecture modules also includes the requirements (requirement statements) attached to the process. Requirement traceability is therefore maintained down to this level. Data flows are allocated to architecture data buses in the same manner. The result is a preliminary design consisting of architecture modules with known functionality, as described by the requirements statements, DFDs, and information notes. This preliminary design also defines the interfaces between modules. In addition to the data flowing across the interface, as defined in the dictionary, the media and protocol are defined in as much detail as possible at this time. The architecture design forms the basis of the top level of integration testing. Once again, requirements traceability and testability is actively addressed and maintained.

During detailed design, the architecture modules are broken down into functional units. Using standard structure charts, the organization of the module is defined as are the interfaces between the units. It should be noted that even though Ada-specific tools are available, the organization of the software can still effectively be described using standard tools and methods.

In the Hatley-Pirbhai method, the "leaves" of the structure chart are defined by Module Specifications (MSPEC). The contents of the MSPEC describes the design of the unit it represents. MSPECs should be approximately one page in length and may consist of text, graphics, or pseudo-code. The MSPEC defines and documents the design for implementation and subsequent maintenance. The structure chart, which defines the organization and interaction of the units of the architecture module, provides the foundation of the lower tier of integration testing. The MSPECs, which define the design of individual units, are the foundation of Unit Testing.

When using Ada, there is some question as to what exactly constitutes a unit. There are three basic approaches when mapping units onto Ada: Unit ≈ Package, Unit = Procedure, Unit = Package/Procedure. Depending on the project, a unit may also consist of multiple packages or a package may consist of multiple units. The different approaches color the entire design and testing process.

A Unit, is defined in both the DoD and NASA arenas as "the smallest testable component" of software. It is therefore the responsibility of the software engineer to define his units small enough so that the software is thoroughly tested yet large enough so that software testing does not become a onerous burden. Another important issue to consider is software reuse. The "unit" is the smallest testable piece software and is therefore the lowest level that is formally documented. This documentation includes functional and interface design details, as well as the testing of same. This can form a natural boundary for the lowest level of a reusable piece of software.

Starting from the external interfaces of an embedded system, it would make sense to configure the software drivers for each interface into individual units. This makes them testable individually, which will speed integration and checkout. This modularity makes the code easily reused for future designs that utilize the same hardware. This modularity also reduces impact to the code when a hardware interface needs to be redesigned.

Moving "inward" in the design, the decisions of what should constitute a unit become more difficult. As with all engineering decision making, it is a series of trade-offs. To properly construct a unit, the same guidelines used earlier in the architecture design can be used. Asking the questions posed by the guidelines is the checklist used to fully understand the problem and gather data necessary to make the "best" choice.

- How closely related are these functions?
- Do they use the same data?
- Is this function used repeatedly by many other modules (units)?
- Have interfaces and data moves minimized?
- Is off-the-shelf or existing software available that performs this function?
- How likely is this function/algorithm likely to change? (a good reason to isolate it!)
- Do the development tools allow visibility into the interfaces for testing?
- Can performance requirements be met by separating this code out?

Properly constructing units is critical from design, test, schedule and budget standpoints. The design must be adequately specified (documented) to allow assurance that it is robust and maintainable. The design must be testable and the testing must be adequate to assure that the software will perform its mission. Each of these steps must add value to the product. Over-testing software won't make it better, just more expensive. This is why the documentation is so important. There must be a clear path from requirements that state WHAT the system is going to do, to the design, which states HOW the system is going to do it. From there, the connection to the code must be clear so the execution of the "HOW" can be verified.

As mentioned earlier, the detailed design consists of structure charts and MSPECs. It can become difficult to relate the structure chart directly to Ada packages and procedures. In fact, it can be difficult to relate the structure chart in the detailed design directly to the code in any language. For this reason, we believe that it is not necessary to generate a source code calling tree for the detailed design. The hardware analog of this idea is that while the schematic represents the design, the assembly drawing, artwork and bill of materials reflects the as-built configuration. Within the structure chart, some of the calls made in the actual source code may be single lines in the pseudo code of the MSPEC and not shown as invocations on the structure chart at all. It may also make sense to group procedures of this type into a common package. In this case, a package could consist of multiple units. However, some of the procedures may be so small that it only makes sense to test them in conjunction with the software that calls them. In this case, not every procedure in a package of multiple units is a unit.

In other cases, multiple leaves of the structure chart may belong in a unit to facilitate testing, yet breaking them out in the detailed design document facilitates understanding. It may also be convenient to develop package specifications for each of these leaves. This would yield a unit consisting of multiple packages.

To be efficient, the engineering process must provide this type of flexibility. One must develop conventions to track the allocation of design elements to code, but allowances for this flexibility are needed to develop value-added documentation that is useful at all phases of the life cycle.

To illustrate the organization and conventions used for design, the following diagrams and text are a brief overview of the process and products used by our organization to specify and design software for NASA and DoD customers.

Requirements analysis starts with the Context Diagram. An example of which is shown in Figure 4. This defines what is "in" the software and what is external to it. This first step is very important since it clearly identifies the scope of the work and the external interfaces. The circle in the center represents the software or system being designed. The rectangles, or terminators, represent the entities that interface with the software. The arrows between the software and terminators represent the data flowing between the two.

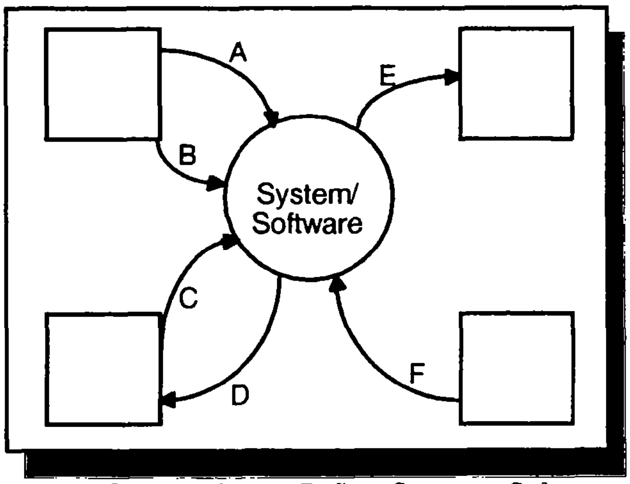

Figure 4. Context Diagram Defines System or Software Boundary

The requirements model is developed in a hierarchical fashion. The level below the context diagram contains Data Flow Diagrams (DFD). Data Flow Diagram 0 is the top most DFD for the software or system. The definition for each data flow is contained in the Data Dictionary. At subsequent levels, the requirements model is developed in greater detail. As the model is developed, checking and balancing is performed to assure that the model is consistent. Interfaces, data flows, and processes must maintain consistency from the context diagram down to the lowest DFD. The Data Dictionary must accurately reflect the flows and their constituent parts as shown in the diagram.

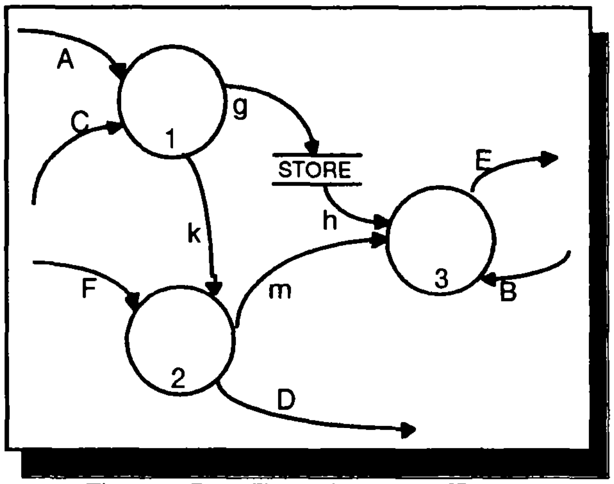

Figure 5. Data Flow Diagram (DFD) 0

The circles in data flow diagrams represent processes. Each process transforms data inputs into data outputs. These processes form the basis for understanding the requirements being developed.

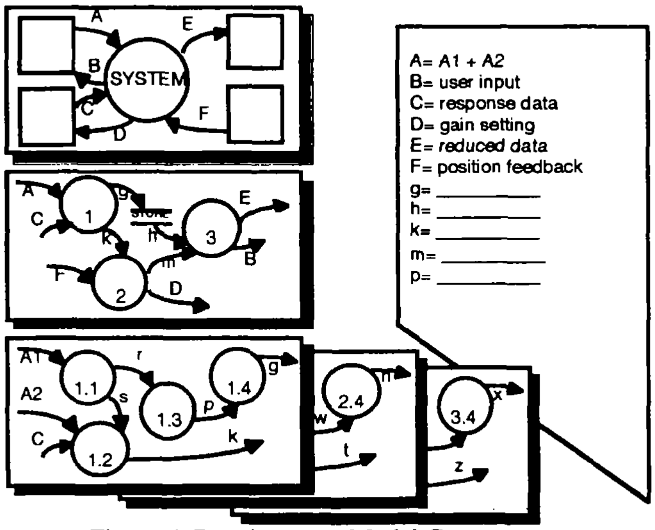

Figure 6. Requirements Model Components

In Figure 6, an overall view of the requirements analysis model is shown. The context diagram is at the top of the hierarchy, followed by DFD 0. Each of the processes shown in DFD 0 is further decomposed into DFDs. These, in turn, may be decomposed further if necessary. The numbering of lower level DFDs and processes tracks their heritage. The Data Dictionary is the repository for data flow definition. When checking and balancing is performed, all data flow entries are reconciled up and down the hierarchy to ensure the model is consistent.The relationship of the requirements model components, the architecture model, and some of the CASE tool conventions used are illustrated in Figure 7. The figure shows the allocation of requirements model processes to architecture modules (identified as Subsystem 1-3 in the diagram). The Architecture Flow Diagram (AFD) forms the top level of the design. Each of these architecture modules is decomposed into design modules or units, as described earlier, which will be described by Module Specifications.

The figure also illustrates the use of notes in the requirements model. There are Information Notes, Requirement Notes, or Traceability Notes. Conventions adopted for note titles indicate the type of note. Within the CASE tool, its point of "attachment" indicates what item it belongs to. Using a combination of commercial and custom software, the requirements document is then produced directly from the model.

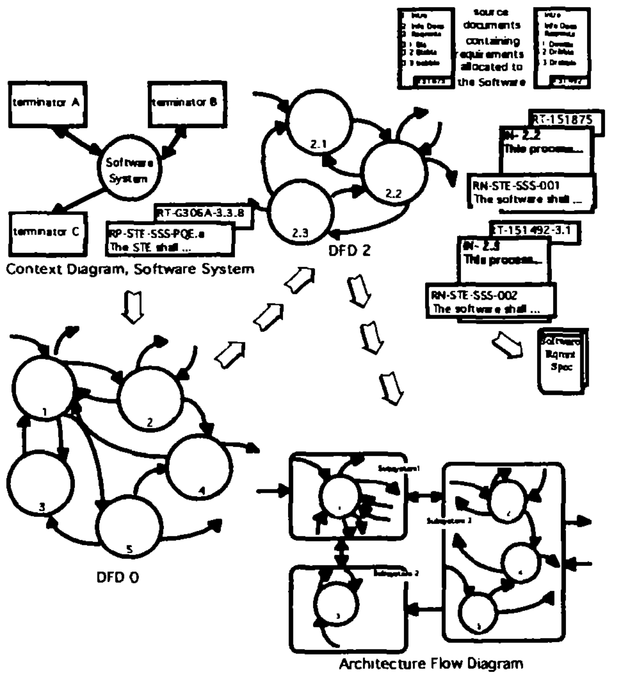

Figure 7. Requirements Analysis and Design Overview

Adopting the Hatley-Pirbhai method has brought discipline to our engineering process. Utilizing CASE tools has enabled the production of high-quality documentation. Together, these tools increase our ability to communicate to our customers and among ourselves by providing visibility into the requirements and design. The completeness of the design is enhanced by this visibility since weaknesses, omissions, and oversights are caught earlier. Perhaps most important, is that the impact of changes is more easily understood allowing a more thorough analysis of requirements and design changes.

APPLICATION EXAMPLE

The application example has been taken from the Tropical Rainfall Measuring Mission's Visible and Infrared Scanner's[7] flight software.

The Visible and Infrared Scanner (VIRS), shown in **Figure** 8, is a cross-track scanning radiometer that is one of 5 instruments aboard the Tropical Rainfall Measuring Mission (TRMM) satellite[8]. The purpose of TRMM is to study the heat distribution and variability of precipitation and latent heat release over several years.

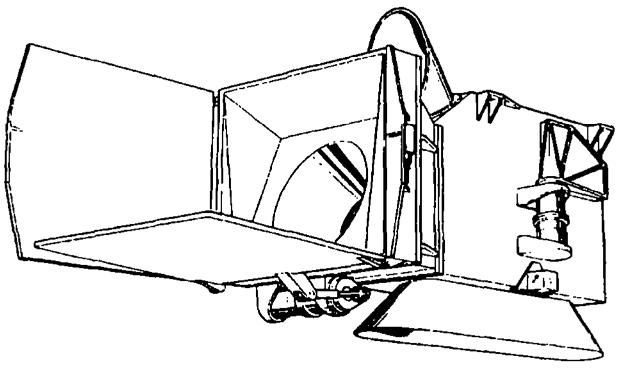

Figure 8. VIRS Scanner Module.

The VIRS instrument supports the TRMM mission by providing data that will be used in conjunction with the Clouds and the Earth's Radiant Energy System to determine cloud radiation. Additionally, once a precipitation system has been characterized using radar, microwave, and other sensor data, it is expected that the VIRS spectral data will be able to be analyzed for precipitation content[2].

The VIRS gathers this data by measuring scene radiance in five spectral bands operating in the visible through the long wave infrared. The VIRS provides a ground resolution of 2.11 km at nadir and a cross-track scan of ±45°. The five spectral bands of the VIRS and their bandwidths are summarized in Table 1.

Band	Wavelength (μm)	Bandwidth (μm)
1	0.63	0.1
2	1.61	0.06
3	3.75	0.38
4	10.8	1.0
5	12.0	1.0

Table 1. VIRS Spectral Bands and Bandwidth

The VIRS instrument consists of two independent modules, a scanner module and an electronics module. These modules are interconnected by a set of shielded electrical cables. The electronics module contains the processor cards as well as the electronics needed to read the data from the detectors (preamplifiers, A/D circuitry).

There are two processor cards, a primary and a backup. Each processor card contains identical flight software. This allows one version of the flight software to be used for the project. The flight software is required to acquire the data from the A/D interface, packetize the data and send it to the Observatory using the MIL-STD-1773 bus. The flight software is also responsible for executing any commands sent to it from the Observatory as well as the acquisition of telemetry and the encapsulation of this information in a housekeeping packet.

Flight Software Documentation- The flight software design process is shown in Figure 9. The left side shows the flowdown of system requirements to the detailed design. The rightmost side shows the flow of implementation from implementation through system test. The two sides are connected through the creation of requirements-based test plans. In general, the left side is implemented using the Hatley-Pirbhai design methodology and the arrows coming from those left-side boxes into the middle boxes indicate different NASA STD 2100 DIDs.

For example, the second row shows the flow from requirements analysis to acceptance test. The requirement analysis was done using a Hatley-Pirbhai structured systems architecture approach. The requirements model that was created formed the basis of the Software Requirements Specification (SRS). Although the SRS was a small document, the mechanics of how it was created dictated that it be rolled out of the Product Specification. In the Product Specification (NASA-DID-P000), under chapter 4 there is a pointer to the rolled out requirements document (NASA-DID-P200). This relationship is shown graphically in Figure 3.

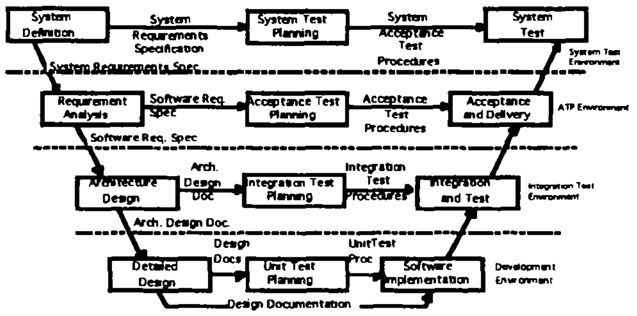

Figure 9. The results of each design process are captured in a NASA 2100 document.

This is in direct contrast to the rest of the documents shown on Figure 9. The system definition as applied to the flight software is captured in chapter 3 of the Product Specification. The architecture design is shown in chapter 5 of the Product Specification. The Detailed Design is shown in chapter 6 of the Product Specification. Using these three chapters and the rolled out requirements analysis, the entire design is captured in one document. This document is only two physical volumes, appropriate for a small project like VIRS.

Flight Software Requirements Analysis- The flight software was designed using a Hatley-Pirbhai structured systems approach. The first step was to perform a requirements analysis. The requirements analysis was driven by several customer documents, system performance specification, and by numerous derived requirements. The derived requirements resulted from the allocation of system functions to the flight software. As is typical with many projects, there were no explicit requirements made upon the flight software. In fact, there were no explicit requirements for flight software at all.

Figure 10 shows a simplified example data flow diagram (DFD) taken from the requirements analysis. This example is a small section of the overall flight software requirements analysis which concentrates on the creation of science and housekeeping packets. These packets are sent to the Observatory from the VIRS and constitute the main data product and instrument health information. A data dictionary, Table 2, shows partial definitions for the data flows present.

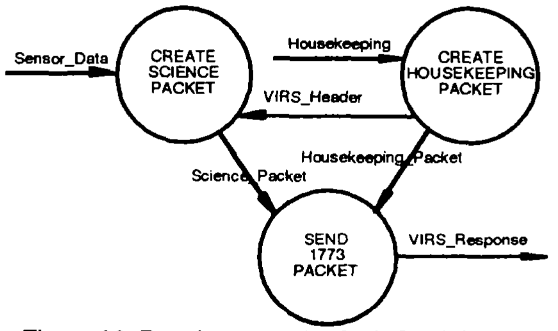

Figure 10. Requirements Analysis Partial DFD

The CREATE HOUSEKEEPING PACKET process creates the Housekeeping_Packet data flow using housekeeping data. Housekeeping data refers to both the analog and digital telemetry. A subset of this telemetry flows from the CREATE HOUSEKEEPING PACKET process to the CREATE SCIENCE PACKET process by means of the VIRS_Header data flow. The Housekeeping_Packet data flows into the SEND 1773 PACKET process.

The CREATE SCIENCE PACKET process is responsible for the creation of science packets out of the sensor data (Sensor_Data) and the VIRS header information (VIRS_Header). The end product of this process is given by the data flow Science_Packet. This flow enters the SEND 1773 PACKET process along with the Housekeeping_Packet flow.

Data Flow Name	Definition
Housekeeping	Analg_Tlm + Dig_Tlm
Housekeeping_Packet	VIRS_Header + Dig_Tlm
Science_Packet	VIRS_Header + Sensor_Data
Sensor_Data	Ch[1..5]
VIRS_Header	Dig_Tlm + BB1 + BB2
VIRS_Response	Science_Packet I Housekeeping_Packet

Table 2. Some Data Dictionary entries.

The SEND 1773 PACKET process creates a Consultative Committee for Space Data Systems (CCSDS) standard packet[1] out of either the Science_Packet data or the Housekeeping_Packet data. Once the CCSDS packet is created, it is transmitted to the Observatory using the 1773 bus.

The relationships between the data flows are succinctly captured in the data dictionary. For example, the VIRS_Response flow is the sum total of all of the possible responses of the VIRS instrument to the Observatory. These responses consist of either the science (Science_Packet) or housekeeping (Housekeeping_Packet) information. In turn, the science packet is defined as consisting of the VIRS header (VIRS_Header) and some sensor data (Sensor_Data). The VIRS header is shown to be a combination of the digital telemetry (Dig_Tlm) and two analog telemetry points (BB1, BB2). All of the data flows have other data flows as their parent or they originate with a terminator block as shown in the previous section on Hatley-Pirbhai.

This information was rolled out of the Product Specification and called the Software Requirements Specification (SRS). There was a pointer to this document placed in chapter 4 of the Product Specification.

Flight Software Architecture- To create the flight software architecture, the requirements analysis data flow diagrams formed the basis of the architecture design. The requirements DFDs were reapportioned to form the basis of the flight software architecture. If we re-examine our requirements example in Figure 10 we can reapportion the requirements to obtain an architectural model. An example of one such reapportionment is shown in Figure 11.

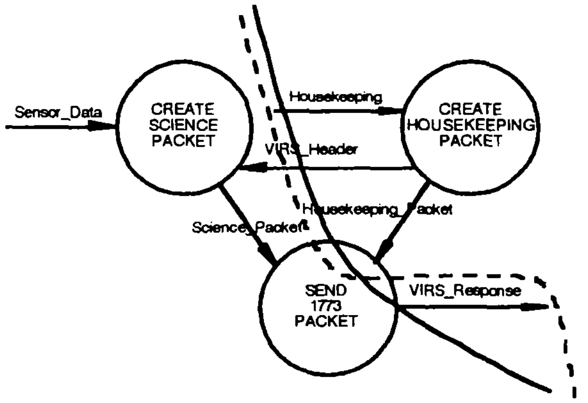

Figure 11. Reapportioning the Requirements DFD

Two architecture modules have been formed from the three requirements analysis processes. The first module, called CREATE SCIENCE PACKET like the requirements derived process, consists of the CREATE SCIENCE PACKET process and a portion of the SEND 1773 PACKET process. This is shown in Figure 11 as a dashed line. A similar process is used to form the other architectural module, CREATE HOUSEKEEPING PACKET. This module consists of portions of the SEND 1773 PACKET process and all of the CREATE HOUSEKEEPING PACKET process. This is shown in Figure 11 as a solid line. Notice that both modules take a portion of the SEND 1773 PACKET. The resultant architecture modules are shown in Figure 12.

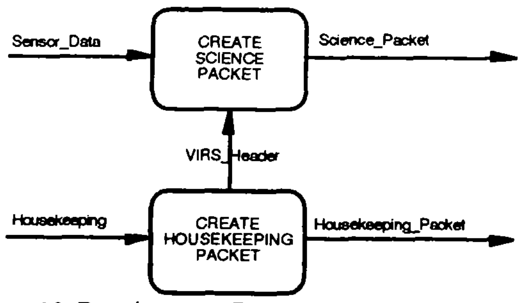

Figure 12. Requirements Derived Architecture Modules

The functionality contained within the SEND 1773 PACKET process has not disappeared, it has merely been absorbed by the remaining modules. From an Ada perspective this is advantageous. The package which implements the mechanics of sending the 1773 packet does not need to know anything about the internals of the CREATE SCIENCE PACKET or CREATE HOUSEKEEPING PACKET. Both of those modules however must know about the SEND 1773 PACKET functions. This allows a non-specific package to be created to output packets to the 1773.

This process is repeated for the entire requirements analysis. The major advantage to creating the architecture out of the requirements model is that you are assured that your architecture has captured all of the requirements. Traceability from architecture to requirement is a fairly straightforward process. The data flow definitions for the architecture were a refinement of the data flow definitions present during the requirements analysis.

The architecture was captured in the Product Specification, chapter 5.

Flight Software Detailed Design- The detailed design took the architecture modules and broke them down into individual units. The relationships between the units is specified using a structure chart. The structure chart shows the invocation sequence of the units. If we take the CREATE HOUSKEEPING PACKET architecture module from Figure 12 and break it down into its structure chart, Figure 13 results.

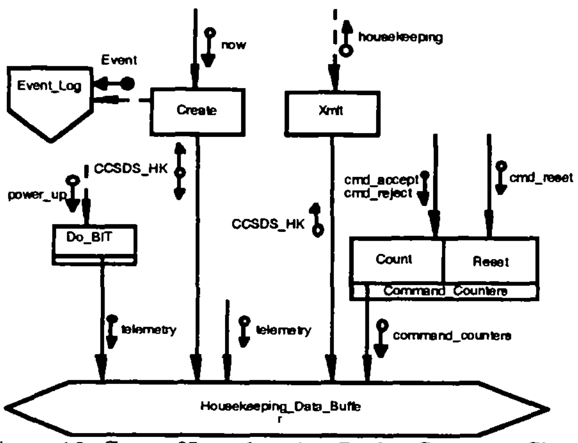

Figure 13. Create Housekeeping Packet Structure Charts

This chart shows an architecture module broken down into four separate units. These units are created due to a functional and architectural grouping. The four units are: The DO_BIT procedure, the EVENT_LOG library procedure (here shown with an off-page connector, the XMIT procedure (shared with the CREATE SCIENCE PACKET architectural module), the remaining three procedures, and a data store.

The DO_BIT unit is a procedure that is invoked by the processor before the Ada environment exists. This procedure is written in 1750 assembly language with the results stored in the 1750's Memory Management Unit (MMU) until it can be retrieved by the housekeeping code written in Ada. This type of procedure requires two interfaces, one assembly and one Ada. The dual nature of this unit precludes it from being included into the CREATE HOUSEKEEPING packet structure.

The EVENT_LOG and XMIT units are library procedures that function like system resources. The EVENT_LOG unit implements the mechanism used to transmit warning or error messages to the TRMM Observatory. Since this facility is used in every almost every architecture module it made sense to break it out as a separate unit. The XMIT procedure is used to transmit data to the TRMM Observatory using the 1773 bus. This procedure was broken out as a unit using reasoning similar to the EVENT_LOG. The only difference is that XMIT is shared among a small group of architecture modules rather than being a ubiquitous resource.

All of the modules shown in Figure 13 are represented by MSPECs. As mentioned earlier, these MSPECs can take the form of text, graphics, or pseudo-code. Ada code is written using these MSPECs as a guideline.

```
INPUT:
  Counter ACCEPT/REJECT

OUTPUT:
  Accept_Count - # Commands Accepted
  Reject_Count - # Commands Rejected

COUNT:
  If Counter is ACCEPT
    Accept_Count := Accept_Count + 1
    If Accept_Count ≥ 2^14 - 1 or
    Accept_Count < 0
      Accept_Count = 0
  else if Counter is REJECT
    Reject_Count := Reject_Count + 1
    If Reject_Count ≥ 2^14 - 1 or
    Reject_Count < 0
      Reject_Count = 0

RESET:
  Accept_Count := 0
  Reject_Count := 0
```

Table 3. Command Counter MSPEC.

Table 3 shows a sample MSPEC for the command counters portion of the CREATE HOUSEKEEPING unit. The command counters count the number of commands accepted and rejected. This data is reported in the housekeeping packet. This MSPEC details the interface of the unit and the algorithms used in the different procedures. Using such a complete breakdown of the individual units eases the coding task.

This portion of the design process is captured in the Detailed Design portion of the Product Specification. The detailed design section can be included in the product specification, as was the case with the VIRS flight software, or it can be rolled out into a separate document using the techniques described in a previous section.

The format of the detailed design section varies widely. One common method is to start with the top level description of the detailed design. This can be facilitated by describing the top level Major Compilation Units (MCUs). MCUs consists of several units grouped together, usually by function. The overall function of the MCU can be described and then a structure chart showing the relationship between the MCU's constituent units is given. This structure chart is then broken down into individual units. These units are explained using a combination of illustrative text and the MSPEC.

Flight Software Test- The tests come directly from the design documents, as shown in Figure 9. The detailed design document is used to create the unit level tests. These tests are written using a white box approach where all execution paths within the unit are traversed. Once the units are working through their respective interfaces the units are integrated together. The basis for this stage of testing is the architecture diagram. The architecture diagram specifies how all of the units fit together. Finally, the entire software project is tested to prove that the software meets its requirements. Those requirements are specified in the Software Requirements Specification (SRS) and form the basis for any and all tests performed upon the software.

MAPPING ADA ONTO HATLEY-PIRBHAI

Mapping Ada onto Hatley-Pirbhai refers to the process of using Ada in the context of the Hatley-Pirbhai methodology. This context only applies to the detailed design level, all higher levels are language independent.

The most common method of combining Ada and Hatley-Pirbhai is to use the Structured Systems Architecture approach to generate the design to the architecture level. From the architecture level on down, Ada specific tools and representations are utilized, most commonly Booch diagrams.

This has the advantage of clearly representing Ada structures, but it removes one of the major advantages of using the Hatley-Pirbhai methodology. That advantage is that the Hatley-Pirbhai methodology introduces a common language for all those involved with the design, from the customers to the engineers.

The approach taken in this paper was to use the Hatley-Pirbhai methodology for the entire detailed design. This makes the relationship between packages and structure charts more difficult to comprehend, as mentioned above, but it has the advantage of explaining the design in a language independent manner.

CONCLUSION

This paper has introduced NASA STD 2100, a tailorable software documentation standard. This standard is tailored by altering the structure of the Software Documentation Set. This set is comprised of four parts: Management Plan; Product Specification; Assurance and Test Procedures; Management, Engineering, and Assurance Reports. There is information associated with each one of these sections.

Each one of these sections consists of two parts: Basic Information and DID specific information. Basic Information is the same for all DIDs and consists of subjects like the Introduction, Related Documents, or Glossary.

The amount of information present in the DID specific information portion tailors the standard for specific projects. Chapters in the DID specific information section can remain as chapters, as is the case for small projects, or they can be "rolled-out" into a separate volume for larger projects.

The Hatley-Pirbhai method for Structured Systems Architecture was introduced as a technology and language independent method for developing software.

Requirements analysis is performed by means of data flow diagrams (DFD) and a data dictionary to define the data flows. These DFDs can have information notes attached to them to facilitate understanding of the design.

The DFDs are broken down into Process Specifications (PSPECs) which define the data transformations that take place within the primitive processes.

Once the requirements analysis is performed an architecture model is created by allocating processes to architecture modules. Because these processes contain the requirements, traceability from requirements to design is maintained.

The architecture modules are broken down into a structure chart. The "leaves" of the structure chart a described by a module specification (MSPEC). The MSPEC can be text, graphics, or pseudo-code. The MSPEC also forms the foundation for Unit Testing.

When used with Ada, the units specified by the structure chart do not necessarily have a one-to-one correspondence with Ada packages. It is preferable to illustrate design highlights with the structure chart than to have an obscure design but with a nice mapping on to Ada.

A breakdown from context diagram to lowest DFD was given.

Following the introduction of the documentation method and the design methodology, an example which employed both of these tools was given. The example given was taken from the Tropical Rainfall Measuring Mission's Visible and Infrared Scanners Flight Software.

This example showed a small sample of the design taken from the requirements analysis all the way to the MSPEC. During this time the method used to document different parts of the design was interleaved into the technical discussion.

The last item shown was how Ada is mapped onto the detailed design portion of the Hatley-Pirbhai structured systems architecture methodology. Most commonly, the Hatley-Pirbhai structured systems architecture methodology is used for the design up to the architecture level with Ada specific tools used for the detailed design.

The approach taken in this paper differed from that by using a strictly Hatley-Pirbhai approach.

In summation, this paper has introduced a NASA software documentation standard which can be tailored to many differently sized projects. It has also introduced the Hatley-Pirbhai design methodology. This was followed by a real-world example of how the documentation standard and the design methodology are used.

ACKNOWLEDGMENTS

The authors would like to thank Mr. Kip Kramer and Mr. Mark Paul at the Santa Barbara Research Center for encouraging the authors to publish this paper. The authors would also like to thank Mr. Joseph Auchter and Ms. Lyn Kawasaki for their comments and feedback.

REFERENCES

[1] Consultative Committee for Space Data Systems. Recommendation for Space Data System Standards - Packet Telemetry, CCSDS 102.0-B-2 Blue Book, Jan 1987.

[2] Goddard Space Flight Center, Tropical Rainfall Measuring Mission Request for Proposal, National Aeronautics and Space Administration, Goddard Space Flight Center GSFC-490-2-01, April 12, 1991.

[3] Guidance On Tailoring and Using the NASA Software Documentation Standard (Draft), System Technology Institute, December 4, 1991.

[4] Hatley, Derek J. Pirbhai, Imtiaz A., Strategies for Real-Time Systems Specification, Dorset House, New York, N.Y., 1987.

[5] Military Standard for Defense System Software Development, DoD-STD-2167A, Department of Defense, October 27, 1987.

[6] NASA Software Documentation Standard Software Management Plan Guidebook, System Technology Institute, November 21, 1991.

[7] Roberts, Dean, Peterson, Brad, Koseluk, Bob, "TRMM VIRS: A Design for Low-Radiation Environments", *1994 IEEE Aerospace Applications Conference Proceedings*, pp. 199~209 February 1994.

[8] Simpson, Joanne, "A Proposed Tropical Rainfall Measuring Mission (TRMM) Satellite", *Bulletin of the American Meteorological Society*, March 1988.

[9] Technical Standards Division, NASA Software Standard Software Engineering Program NASA-STD-2100-91, National Aeronautics and Space Administration, July 1991.