REQUIREMENTS DEFINITION FOR ADA-BASED TRAINING SYSTEMS

by

Mr. Mike Caffey
and
Dr. Matt Narotam

Burtek, Inc.
Tulsa, Oklahoma

ABSTRACT

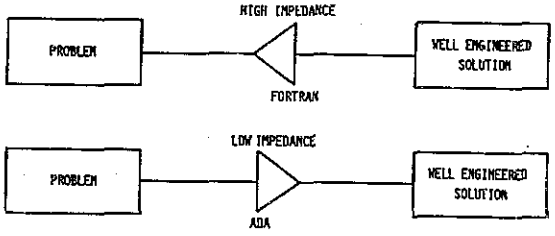
The importance of the requirements definition stage in developing Ada for simulator systems is one of the "lessons learned" on the Ada Simulator Validation Program (ASVP). The traditional approach to requirements definition generally utilized for training systems is reviewed and some of the problems that result are discussed. The types of requirements that impact the design and life cycle support of the system are defined because of their significance to the process utilized for developing the system design. The impact that Ada and Object-Oriented Design implementations have on the requirements definition process is examined by first addressing the characteristics and features of Ada that satisfy software engineering concepts. Next, the decomposition and design procedures of the method and the manner in which requirements are utilized for generating the software design are discussed. The process involved in the establishment of these requirements is also discussed. Finally, activities related to the systems requirements review process are addressed.

1.0 INTRODUCTION

Ada was developed to support the engineering approach to software development. Ada features such as tasking, packaging, generics, English-like syntax, strong typing, exception handling, etc., support software engineering concepts such as abstraction, information hiding, modularization, and generalization. The appropriate use of these features will lead to well-engineered software systems that are reliable, easily maintainable, highly reusable, and efficient. Languages such as FORTRAN provide very few features that support software engineering. This has the effect of "distancing" the problem from the desired solution. Traditional software engineering practices that are used for developing FORTRAN software therefore do not support a mapping of the program solution to the problem domain. The diagram below attempts to demonstrate that FORTRAN provides a high impedance path, whereas Ada provides a low impedance path to developing well-engineered solutions.

Ada, therefore, has an impact on traditional software engineering.

Having developed a language that will support the engineering of a solution that will map onto the problem domain, a process or method is required to handle the transition of the problem onto a solution. Burtek has refined the Object-Oriented Design process to manage this transition.[1] This process, called Object-Oriented Development[1], employs the concepts of abstraction, information hiding, modularization, and generalization to convert problem requirements into a solution.



Object-Oriented Development possesses certain inherent characteristics which affect requirements definition. Both Object-Oriented Development and Ada place great emphasis on the requirements of the systems to be developed. This is consistent with a systems engineering approach to the development of training devices. The specification of requirements, therefore, becomes even more critical because of its impact on software design and performance. This paper addresses the impact of Ada on this process based on experience gained from the ASVP. The ASVP is a research and development contract issued by and sponsored by the USAF Aeronautical Systems Division (AFSC), Wright-Patterson AFB, Ohio. The ASVP involves the redevelopment of software for demonstration on a C-141B Operational Flight Trainer (OFT). The development effort is focused on gathering engineering design and acquisition data that highlights the effectiveness of software engineering concepts permitted by the Ada language. Part 2 of the paper briefly describes the traditional requirements definition process and the problems resulting from this process. It will be seen that this process, which influences the development of the software system, does not support Ada software development using software engineering concepts. Part 3 describes the impact of requirements on the establishment of the system design using the Object-Oriented Development method and Ada. Part 4 describes the requirements derivation process, beginning with user requirements. It is important to note that the user requirements comply with the characteristics established in Part 3. Part 5 addresses the review process and considerations which impact the review of requirements and their inclusion in the system design. Note that in the discussion that follows, the term component represents an object.

2.0 TRADITIONAL REQUIREMENTS DEFINITION PROCESS

Figure 1 depicts the traditional process for software development. Four major steps are identified beginning with requirements definition and ending with code, integration and test. The requirements definition step leads to the establishment of a data base of systems requirements. These requirements are derived from

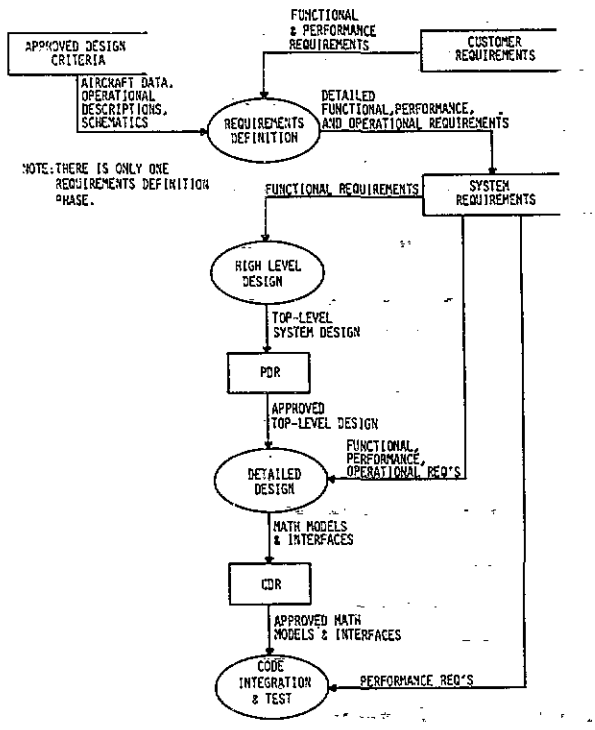


Figure 1. Traditional Requirements Definition

an approved design criteria for the system being simulated and customer requirements. The customer requirements are mainly incomplete functional and performance requirements. The systems requirements are refined and elaborated by the manufacturer.

The intermediate step results in the development of the high level design followed by detailed design of the system. There are typically two major design review phases - Preliminary Design Review (PDR) for reviewing the high level design, and Critical Design Review (CDR) for reviewing the detailed design of the system.

Traditionally, the design reviews tend to focus on implementation details instead of considering whether performance requirements have been established and that these requirements are being met. This focus on implementation detail begins at the Request for Proposal (RFP) stages where often the specification of the "system requirements" is in the form of a preliminary breakdown of system functions which tend to form the basis of the work structures and breakdown. That, together with a "pre-set" mentality of what the customer wants (a cockpit procedures trainer or an operational flight trainer or a weapons system trainer) appear to be the prerequisite for establishing the system requirements. This process tends to lead to the definition of the design using a functional approach. The functional approach elaborates the subsystem definition based on available information about functional requirements. This approach leads to ill-defined components and interfaces, resulting in software not easily maintained or reused, and that which requires several changes to account for omitted requirements during the design phase.

The traditional requirements definition process gravitates toward a specification of implementation concerns rather than focusing on a better definition of the problem scenario. This process requires modification to accommodate the procedures required to support Ada software development. The changes impact the process of requirements derivation, the types of requirements, and the characteristics of these requirements.

3.0 Ada AND DEVELOPMENT METHODS

Ada has been designed to solve many problems inherent in older programming languages. Ada constructs allow the system architecture to be more closely tied to requirements. Used appropriately, cost effective systems can be developed that are reliable, portable, and maintainable.

One major feature of the language is that software specifications can be developed and integrated to form the software structure, prior to coding. This feature supports the recursive decomposition of system into subcomponents, with well defined interfaces that can be used to generate Ada specifications. This process, supported by a Program Design Language (PDL) Tool, can provide a powerful mechanism for development of well structured and well designed software systems because it allows prototyping as the software is composed.

Object-Oriented Development best suits Ada as a development method. The packaging and tasking constructs of Ada allow the system to be partitioned into logical structures. This feature supports the process of Object-Oriented Development for generating the software structure. Object-Oriented Development allows the breakdown of the system into objects that act on other objects within the system. These objects can be implemented in Ada to develop real-time software systems. The object content of the system is determined from requirements imposed on the system and the approved system design criteria.

Figure 2 depicts the main steps in Object-Oriented Development. As shown, Object-Oriented Development is a recursive process that generates the structure of the software system. Steps 2 and 3 are recursed for "complex" subcomponents until the lowest level components are identified. Figure 3 provides a more detailed view of decomposition process. The process utilizes top level requirements to define system components and actions performed on the components. Additional requirements called "derived requirements" are generated for defining lower level components and their interactions.

The use of Object-Oriented Development for Ada software development requires design requirements to be hierarchically derived from the consideration of user requirements and the system design criteria. Top level requirements will be rather general, while lower level requirements will be somewhat more specific. Requirements pertaining to a particular component should specify only the external characteristics of the component, leaving the internal design of the component to the engineer. At the lowest level, the requirements should completely define the characteristics of the component behavior.

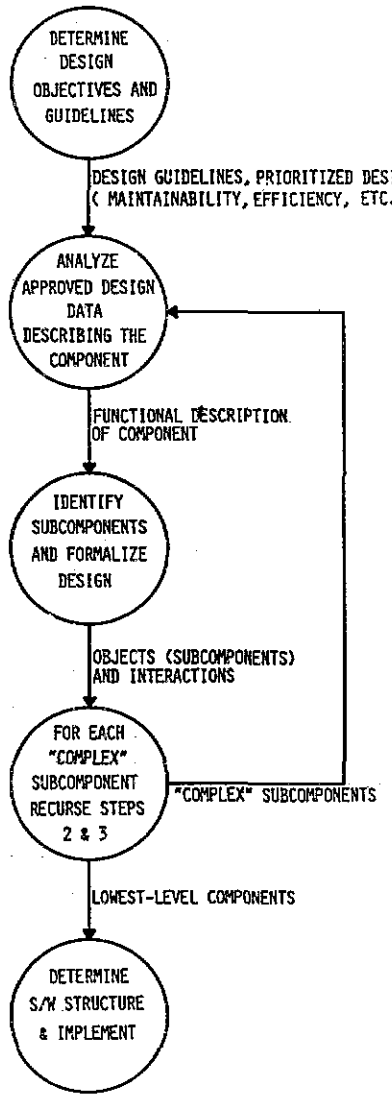


Figure 2. Main Steps in Object-Oriented Development

When specifying requirements for a real-time system, the engineer should specifically define the following:

1. Activities that should appear to be continuous or concurrent with other activities.

2. The response of the component to undesirable events.

3. Activities that are time dependent vs. activities that are purely event driven.

4. Fidelity requirements such as screen resolutions and output tolerances.

5. Required execution speed and allowable memory space.

In addition, to address the life-cycle of the product, the requirements should specify a prioritized set of design guidelines addressing maintainability, reusability, portability, and run time efficiency, as well as outlining forceable enhancements to the product. A set of implementation priorities should be specified and the environment in which the product is to operate should be characterized.

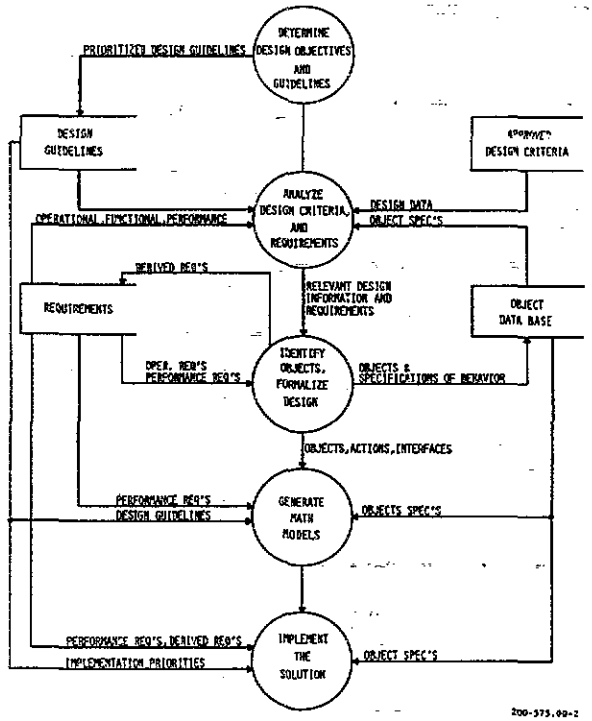


Figure 3. Detailed Decomposition Process

4.0 REQUIREMENTS DEFINITION PROCESS

Figure 4 depicts the requirements definition process embedded in Object-Oriented Development. The initial requirements, combined with the approved design criteria, are used for establishing a description of the system specification in terms of components and actions using abstraction and information hiding. This initial set of requirements must be correct and sufficient to allow the definition of the system specification. These requirements are generally established by the customer by examining the training requirements of the simulator. It is essential that the customer understand the impact of these requirements on the design process. Good definition of requirements will lead to a well designed software system that is easily maintainable and usable (and reusable).

At each decomposition step, additional requirements are derived which result from consideration of limitations, assumptions, and additional design detail. The performance characteristics and fidelity of implementation will impose requirements on the system design. Constraints to the system design may be imposed by hardware or other components.

The derived requirements are utilized for generalizing low level system/subsystem specifications. These specifications form the basis for determination of components, actions, or operations to be performed by the components. This process is repeated until the system is fully defined, based on the elaboration of requirements.

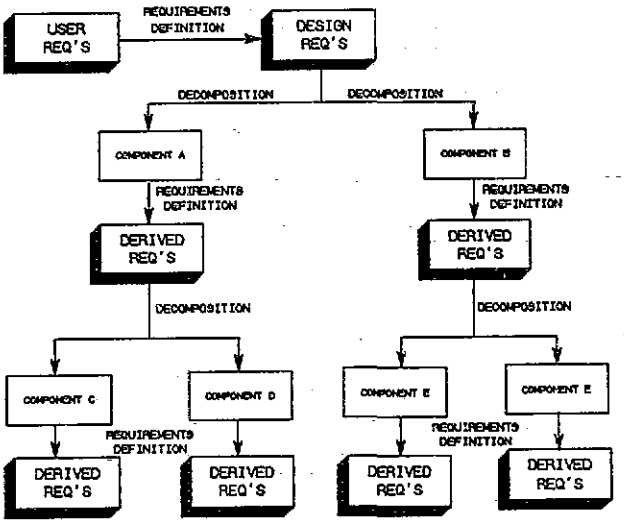


Figure 4. Requirements Definition Process

5.0 REQUIREMENTS REVIEW PROCESS

The use of Object-Oriented Development and Ada will impact the review processes during the development phases of the project. Requirement review should take place frequently until the design is complete. Working reviews should replace formal reviews. The customer must be actively involved in the review process with the objective of "working together" towards system implementation. This approach to software development results in the synthesis of the system[2], and yields all requirements, constraints, limitations, and assumptions which must form part of the life cycle support documentation. By using an appropriate design tool, the system design can be released incrementally for review and acceptance, to form the baseline for production of software. The customer can also be involved in this process, which will result in a more cost effective review and verification process.

6.0 CONCLUSION

This paper has attempted to describe the importance of requirements definition to the software development process. The process has been enhanced because of the need to apply a development method for generating Ada software. Enhancement to the process will result in a closer mapping of the software solution to the training requirements. This is dependent on well defined requirements for the training system. This puts a burden on the customer to train personnel to understand the impact of these requirements on system definition.

REFERENCES

1. Narotam, Saib, Layton; Object-Oriented Development of Training Systems in Ada, Proceedings Nov 18-20 1986, Interservice/Industry Training System Conferences, Volume 1, Page 13.

2. Hoog; Systems Engineering For Training Systems - A Team Approach, Proceedings, Nov 18-20, 1986, Interservice/Industry Training Systems Conference, Volume 1, Page 190.

ABOUT THE AUTHORS

Mr. Mike Caffey is a software systems engineer at Burtek where he develops simulation models for on board aircraft systems. Mr. Caffey holds a degree in Electrical Engineering from the University of Tulsa. While involved in the Ada Simulator Validation Program, Mr. Caffey has performed extensive research in design methodologies and tools that are used for the development of real-time software in Ada. Mr. Caffey is currently involved in the use of object-oriented design for the redevelopment of the aircraft systems software for a C-141 Operational Flight Trainer.

Dr. Matt Narotam is a staff engineer at Burtek, responsible for the technical performance of the Ada Simulator Validation Program. Dr. Narotam holds a Ph.D from the University of Salford, England, for a Thesis on Continuous Systems Simulation Language (CSSLs) implementation on mini-computers. Prior to joining Burtek, Dr. Narotam held the position of Research Scientist at the Computer Simulation Center, University of Salford, where he investigated simulation techniques for CSSLs. At Burtek, Dr. Narotam led a team of systems engineers in the development of training equipment for the F/A-18 aircraft. Prior to his appointment as staff engineer, Dr. Narotam was Supervisor, Software Systems. Dr. Narotam is an Adjunct Professor of Software Engineering at the Maths and Computer Science Department, University of Tulsa.