N89-16299

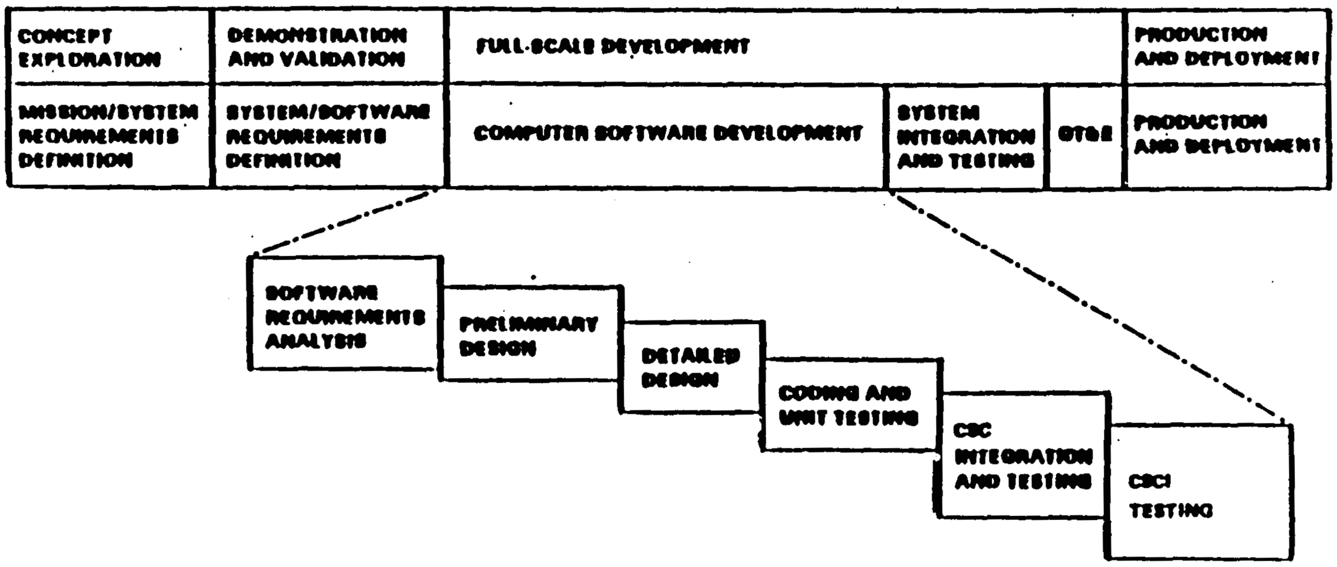
S20-61
167044
10P.

Analysis and Specification Tools in Relation to the APSE

John W. Hendricks

Systems Technology, Inc.

Ada and the Ada Programming Support Environment (APSE) specifically address the phases of the system/software lifecycle which follow after the user's problem has been translated into system and software development specifications. The "waterfall" model of the lifecycle identifies the analysis and requirements definition phases (now known as the concept exploration and the demonstration & validation phases in the lifecycle as described in the new DOD-STD-2167) as preceeding program design and coding.


CONCEPT EXPLORATION
DEMONSTRATION AND VALIDATION
FULL-SCALE DEVELOPMENT
PRODUCTION AND DEPLOYMENT
MISSION/SYSTEM REQUIREMENTS DEFINITION
SYSTEM/SOFTWARE REQUIREMENTS DEFINITION
COMPUTER SOFTWARE DEVELOPMENT
SYSTEM INTEGRATION AND TESTING
OT&E
PRODUCTION AND DEPLOYMENT
SOFTWARE REQUIREMENTS ANALYSIS
PRELIMINARY DESIGN
DETAILED DESIGN
CODING AND UNIT TESTING
CSC INTEGRATION AND TESTING
CSCI TESTING

Since Ada is a <u>programming</u> language and the APSE is a <u>programming</u> support environment, they are primarily targeted to support program (code) development, testing, maintenance, etc. The use of Ada based or Ada related specification languages (SLs) and program design languages (PDLs) can extend the use of Ada back into the software design phases of the life cycle (for example, see Goldsack). However, there seems to be some agreement that Ada is not appropriate as a language for dealing with the "problem space" and the earliest phases of the lifecycle (Brodie, Mylopoulos, and Schmidt, p.410; Booch,p. 359).

The Ada Programming Support Environment (APSE), and indeed the Ada language itself, was defined as a response to the "software crisis" in DOD embedded systems. Booch (p.7-8) lists a number of symptoms of this situation, including:

- o <u>Responsiveness</u>. Computer-based systems often do not meet user needs.

- o <u>Modifiability</u>. Software maintenance is complex, costly, and error prone.

In particular, software maintenance is identified as being responsible for between 40% and 70% of the total hardware and software expenditures for these systems. We can expect that many of the systems for the NASA space station will share important characteristics with the DOD embedded systems (e.g., complexity, long-lifetime, changing requirements, real-time interfaces), and they should be subject to many of these same problems.

The world's best programming effort can not produce a system which is responsive to the user's needs if the requirements upon which it depends do not describe an appropriate solution to the user's problem or if the requirements are in a form which we have great difficulty translating into an implementable design. Also, if this problem exists with the original requirements for a system, it can be repeated every time there is a change in the problem. We do not have data which characterize the distribution of software maintenance costs between "bug fixes" and changes in requirements, but it would not be surprising if a large part of the "maintenance" costs are caused by evolution of the requirements, especially for systems which are in service for a number of years. Therefore, both the responsiveness problems and a large part of the maintainability problems which characterize the software crisis may be beyond the reach of Ada and the APSE, unless facilities to deal with the processes of concept exploration and demonstration & validation can smoothly be linked into the APSE.

There are a number of developments which demonstrate the feasibility and desirability of formalizing specifications or architecture designs at higher levels of abstraction than that provided by a programming language (e.g., Balzer; Zave). These efforts share an objective of reaching out toward the "problem space" with a representation which is much easier to use than a programming language for describing the requirements, but is still capable of being translated or transformed into compilable code with limited manual intervention (automatic programming). They also share a commitment to extensive use of computer based tools to support the processes of analysis, specification and design. To the degree that these approaches succeed, they can address the problems of responsiveness to initial user needs and maintenance of responsiveness as these needs change over the lifetime of the system.

It is unlikely that any of these efforts will eliminate the need for substantial amounts of human programming in the development of the large and complex systems for which Ada and the APSE are designed. If these new techniques are to be exploited for major projects such as the N/.SA space station, they must be capable of being used in conjunction with program design and development under the APSE.

One of the most promising of these new systems is Process Architecture Design Technology (PADtech). Systems Technology is working with the developers of this system, Associative Design Technology, Ltd (USA), to introduce and support this new technology for aerospace and military applications. An overview of PADtech and some of the issues raised by its use with the APSE should suggest both the promise of these new systems and some of the issues to be considered in "integrating" these new tools into major projects which will be using the APSE.

PADtech includes both a methodology and a set of computer based tools to support the use of the methodology in creating an architecture design for a complex system. The methodology provides a representation to formally describe:

- o the structure of processes which we expect the system to implement, the events which will cause each process to be executed, and the events which each process can cause to occur; and

- o the conceptual structure of the entities involved in the processes in terms of the role relation hips between the concepts, object types and objects.

This representation (Process Architecture Design specification Language or PADL) describes processes which may be implemented by hardware, or by persons following procedures, as well as by software. However, PADL has a precise semantics which enables it to be transformed into executable forms, and this inevitability makes its application a more demanding process. By way of contrast, Goldsack (p.11) noted that "...the ease of use of PSL, SADT and many others, is partially due to the absence of a precise...semantics."

The computer based tools for the application of PADtech include the following:

- A design workbench which provides a high performance, color, icon driven, interactive graphics interface for the creation and manipulation of the graphical form of the Process Architecture Design specification Language. The design workbench supports the system architect in the evolutionary process of analysis, specification and design. It also provides support for interactions with problem area experts and with program designers and programmers.

- Modules which translate between the graphical form and the textual form of the Process Architecture Design specification Language;

- A data manager which provides bookkeeping support for the evolving process architecture design;

- A facility for building up a customized set of icons, process models, etc. which are appropriate for specific problem areas.

- An interpreter for simulated execution of the process architecture for an early prototyping, iterative design cycle.

- A "monitor" which collects the results of the interpretive execution.

- A "debug" environment for controlling and examining the results of interpretive execution.

- Code generation facilities for transforming Process Architecture Design specification Language descriptions for process and conceptual structures into the implementation languages, Ada and SQL.

PADtech is designed to be applicable to the analysis, specification and design process at several different levels. First, it can be used at the strategic planning level. For example, one can build a process architecture to represent an entire organization or a major project, and use this "enterprise model" to identify and specify automated information and communication systems to support operation of the entire enterprise. Second, PADtech can be used at the system or integration architecture level for a specific system. It can be used to design the architecture which defines the overall structure for a complete system, or to design and implement a database and communication "substrate" to integrate many separately developed modules, including man- and hardware-in-the-loop elements. Third, PADtech can be used to specify, design and implement (by code generation) systems which can readily be characterized by "object processing" processes, i.e., processes which create and change the state of both abstract and "real" objects.

PADtech will be most beneficial when applied to systems with some of the following characteristics:

- o Requirements which are complex, not completely understood, and are expected to evolve over the life of the system,
- o Requirements for very high speed execution involving parallel and/or distributed execution,
- o Requirements for real-time responsiveness,
- o A requirement for high speed management of complex, interactive data bases and communication structures,
- o Integration of a large number of processes while maintaining protection against catastrophic failures.

We expect that there will be a number of space station systems with these characteristics, and that PADtech and other innovative tools for analysis, specification and design will be required to make these systems responsive to the requirements and maintainable over a long lifetime.

What are some of the issues raised by the use of these tools with Ada and the APSE? First, tools which are geared to creating a problem space oriented, executable specification or design specification tend to cut across the phases of the lifecycle as defined in the waterfall model. These tools gain much of their utility from an iterative cycle of analysis, execution and evaluation of the specification as a "prototype," re-analysis, etc. They emphasize direct involvement of the users or problem area experts in evaluating the implications of a design specification as they are revealed by repeated prototyping. The analysis and prototyping processes are supported by an interactive environment which is heavily dependent on "prototype execution" and graphics for presentation and manipulation. Also, these new techniques push formalization back toward the problem specification and use (partially) automated transformation to generate code modules. This allows maintenance which is occasioned by changes in the requirements, to be performed on the specification/design rather than on the code. Then, the revised specification is transformed into updated code modules. Use of these new techniques will be made more difficult if a rigid segmentation into the phases of a waterfall lifecycle model is imposed by procurement processes or by implementations of the APSE.

Second, there are several reasons why specification and design tools should be linked into the APSE. Most importantly, if design specifications such as those in PADL are to be used for maintenance and are to become a part of the permanent documentation of a system, it is important to have control over their versions as one does for code modules. Also, in spite of everyone's tendency to claim that his system is complete and universal, none are. All of the analysis, specification and design tools would benefit from being able to interface with other systems which could complement their own capabilities (for example, see Ripken). An "open" APSE could coordinate between several "outside" tools, as well as between these tools and code development under the APSE.

Third, the amount of effort being put into the development of Ada and the APSE creates a certain momentum towards making them all inclusive. If Ada is the programming language, why not use it as the basis for a design language, a specification language, a conceptual design language, etc., and mandate their use? If the APSE is to control the programming process, why not mandate that only tools which are fully integrated into the APSE can be used from concept exploration onwards? The potential benefits of such a coherent, start-to-finish development environment need to be balanced against the potential costs of using much less than optimal tools in the pre-programming phases of the lifecycle.

A detailed examination of these issues would be a major project and is not contemplated here. However, we will suggest that in applying Ada, the APSE and standards such as 2167, we should be careful not to let their application expand to a point where they stifle innovation. The continuing revolution in microelectronics is providing an opportunity to create systems to solve increasingly complex problems; new techniques for specification and design will also be needed to exploit this opportunity. Many of these new techniques, which will be needed to build the increasingly complex systems we require, will not be developed exclusively for use by one industry or one language. Retaining the option to select different methodologies for problems which have differing characteristics may be the only effective approach at this time.

Recall that the standardization of the APSE as a programming support environment is only now happening after many years of evolutionary experience with diverse sets of programming support tools. Restricting consideration to one, or even a few chosen specification and design tools, could be a real mistake for an organization or a major project such as the space station, which will need to deal with an increasingly complex level of system problems. To require that everything be Ada-like, be implemented in Ada, run directly under the APSE, and fit into a rigid waterfall model of the lifecycle would turn a promising support environment into a straight jacket for progress.

References

Balzer, R., "A 15 Year Perspective on Automatic Programming," IEEE Trans. Software Eng., vol. SE-11, pp. 1257-1268, November 1985.

Booch, G., Software Engineering with Ada, Menlo Park, CA: Benjamin/Cummings, 1983.

Brodie, M., J. Mylopoulos and J. Schmidt (eds.), On Conceptual Modeling; Perspectives from Artificial Intelligence, Databases and Programming Languages, New York: Springer-Verlag, 1984.

Goldsack, S. (ed.), Ada for Specification: Possibilities and Limitations, Cambridge, England: Cambridge University Press, 1985.

Pepper, P. (ed.), Program Transformation and Programming Environments, Berlin: Springer-Verlag, 1984.

Zave, P., "The Operational Versus the Conventional Approach to Software Development," Commun. ACM, vol. 27, pp.104-118, Feb. 1984.