DESIGN, DEVELOPMENT AND MANAGEMENT OF REUSABLE SOFTWARE COMPONENTS IN ADA*

JAMES O'DAY
SPERRY SYSTEMS MANAGEMENT
SIMULATION SYSTEMS
RESTON, VIRGINIA

ABSTRACT

The use of Ada and reusable software components in flight training devices promises to significantly reduce the cost and development time of future trainers. This paper deals with the selection, design, development, and management of reusable software components utilizing Ada as the design and implementation language. In order to realize the benefits of reusable software components, careful planning is needed to ensure that appropriate candidates are selected. In this selection process the additional cost of producing an efficient real time software component is discussed and weighted against the useful lifetime of such a component. Since design is the most critical part of the development cycle of a reusable component, generality versus efficiency tradeoffs are discussed in terms of how they will affect the future success of the component. The successful management of reusable components and their acceptance by programmers is also discussed.

INTRODUCTION

The cost and complexity of flight simulation devices is rapidly escalating along with the need for cheaper, more realistic training. Since a flight simulator is so software intensive, the software development process has a major impact on cost and schedule. Flight simulators are becoming more software intensive in an effort to keep up with the increasing amounts of information being processed on board modern aircraft as well as to simulate modern threat environments. Reusable software is one means by which software development costs can be reduced.

Reusability as it is used in this paper refers to the ability to reuse previously developed software components with little or no modification necessary. For the purposes of this paper, a software component is defined to be an Ada procedure, function, package, or generic. The scope of reuse possible includes system, subsystem, or individual elements. Reusability of software components is not a new concept but has been around for quite some time; the most common example being math libraries. The amount of software reused has varied from company to company, and industry to industry. The Japanese have successfully applied this concept in their culture to produce "Software Factories" where programmer productivity has greatly increased because of the amount of reuse. Reuse levels of 50 - 60 percent have been reported in some types of business applications. While a feasible level of reuse in real time flight simulation applications has yet to be determined, the following can be assumed to be typical benefits of reuse.

- Reduced software development time
- Reduced software development and maintenance costs
- More thoroughly tested and reliable software components
- Enhanced rapid prototyping capability

The level of software reuse is directly related to several factors which will be discussed in more detail later. These include:

- Application domain requirements (especially efficiency)
- Experience level of software designers
- Programming language
- Management commitment to reuse
- Organizational structure to support reuse
- Documentation
- Programmer acceptance
- Availability of other reusable components

Three major tasks are associated with producing and applying reusable software components to any application. These are:

1. Identification and selection
2. Design and development
3. Management

IDENTIFICATION AND SELECTION

All the major areas of flight simulator software development should be considered when identifying and selecting candidates for reuse. In a typical flight simulator the major areas of software development not including special purpose devices such as visual and radar landmass systems are:

* Ada is a Registered Trademark of the U S Government (Ada Joint Program Office)

- Flight dynamics
- Engines
- Aircraft systems
- Instructor operator station (IOS)
- Real time system software
- Software tools
- Atmospheric environment
- Tactical environment
- Aural and motion cueing

In choosing whether to design and implement a software component as a reusable component, several factors must be taken into account. The first and most important of these is efficiency. Since reusable components tend to be more general than software components written to satisfy a single requirement, the reusable component is likely to be less efficient in a real time environment than a non-reusable component. The danger here is that money saved on software development would not compensate for the additional hardware expense required to support the reusable component at the high iteration rates characteristic of some flight simulation modules such as flight controls. This, in essence, points to the need for a cost tradeoff analysis which would be used to weigh several factors before a decision is made on whether a software component is a good risk or not. The most difficult part of such an analysis is how to quantify savings from reusable components, since reusable components are likely to require some modification to be usable from one simulator to another. Figure 1 represents general relationships between the cost of modifying reusable software versus the cost of original development. Reuse becomes questionable as the amount of modification to a reusable software component increases. The greatest cost savings come with the least amount of modification. A reusable

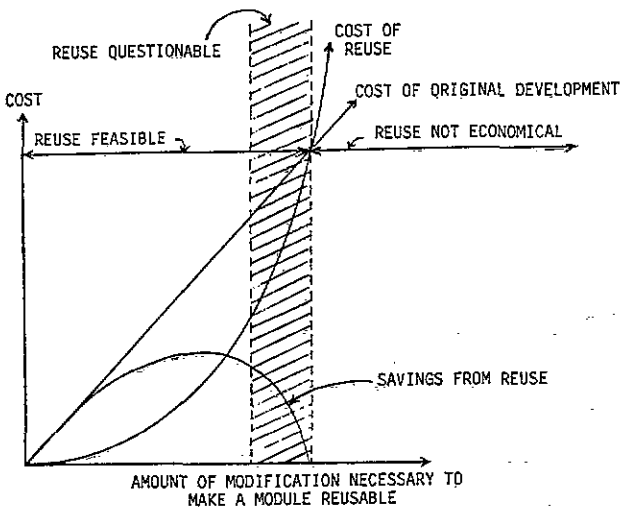

FIGURE 1

software component can end up costing more than original development if the amount of modification becomes extensive.

Another major area of concern in selecting reusable candidates is the useful lifetime or payback period of the software component. Some simulator software areas are more technologically stable than others, which means that the added costs of developing a reusable component would be offset by its useful lifetime. Some aircraft systems such as avionics are constantly being updated. Good candidates in such an area would be systems which have been standardized within one or more of the military services. This includes software associated with such systems as the Global Positioning System, 1553 avionics bus control, Standard Air Data Computer, Standard Inertial Measuring Unit, etc. Bad candidates are systems based on older technology such as mechanically stabilized inertial units as opposed to newer computationally stabilized strap-down systems. Bad candidates include systems whose future use is very limited or doubtful and which will not provide an adequate return on investment.

Another area where technology impacts flight simulators is in engines. Because of the high cost of developing engines, some engines such as the J-79, F-100 and F-404 are common to several different types of fighter aircraft making them good candidates for reusable software. Other good candidates based on similar criteria would be:

- Magnetic variation
- Radio facilities database management
- Atmospheric environmental effects
- General IOS utility functions
- Math libraries
- General purpose system programming utilities
- Motion and G-suit/seat cueing
- Tactical threat environment (if table/data driven)

DESIGN AND DEVELOPMENT

One lesson we have learned about reusing software is that reuse must be designed in from the beginning. Reusable software components do not just happen--they must be planned. In comparing the design and development of reusable software components to the way software is normally developed for a flight simulator, several differences are readily apparent. Although both require traditional forms of requirements analysis and development of detailed specifications, the requirements for reusable software components are significantly more demanding than those for conventional flight simulation software design. This is due in part to the fact that the software component must be more generic in order to respond to differing requirements from simulator to simulator while at the same time being efficient enough to meet real time requirements. This also makes the job of specifying what the component

must do more difficult. If the specifications are too demanding, efficiency and real time performance will suffer. On the other hand if the specifications are not demanding enough, the component will not be suitable for reuse. While these problems are not unique to reusable software components, they are more demanding tasks than we have faced in the past. Therefore it is logical to assume that more time and effort are going to be required in this part of the development process. This will make the cost of the reusable component more expensive initially and will lengthen its development cycle. It should also be anticipated that the first versions of the software component that are produced will not completely satisfy the requirements and that the design will go through several iterations and will be a compromise between efficiency and functionality. Care must also be exercised so that a component is not over designed. The more reusability desired, the more difficult and complex the design task becomes, reducing the likelihood of its success as a reusable component.

Ada is a much better language for supporting reusability in flight simulation than Fortran 77 because the language has been designed with reusability in mind. Ada supports the goals and principles of software engineering much better than Fortran 77 (7). Capabilities of Ada such as strong typing, packages, generics, information hiding, exception handling, and separate compilation are features which support reusable software but were not available in Fortran 77. Table/data driven forms of reusable software which worked well with Fortran work equally well with Ada.

Packages and generics are two of the most important features of Ada for creating reusable software components. Packages provide a means of structuring reusable components into logically related units. For example, flight simulators all contain some type of attitude indicator. While there are many different types of attitude indicators, they all share some common functions such as pitch and roll. In creating a reusable software component for such a device, packages give us the capability to isolate the core functions common to all aircraft simulators from those that are device specific. This means that pitch and roll functions could be handled in one procedure, turn and slip indications in another, navigation and guidance functions in another, and so on. The Ada package lets us group these procedures into one logically related unit while the generic construct lets us create more than one instance of such an object; in this case pilot, copilot, and standby attitude indicators. This type of package could then be reused on another flight simulator by modifying those procedures which represent new or different functions of the attitude indicator without altering the core functions of the basic attitude indicator.

A major goal in designing reusable software components is hardware and software independence. This will become increasingly important in flight simulation as we begin to move into distributed processing environments. If we are to be able to transport a software component such as an engine from a basically sequential frame-driven supermini computer environment into a parallel processing environment, partitioning of the component and its communication interfaces must be carefully considered to prevent hardware and software dependencies from entering the design and hampering its reusability.

MANAGEMENT

If reusable software components are to reduce software development costs, they must be properly managed within a company. They are a type of resource that is wasted if left unused. The management of a company must make a commitment to reuse not only in terms of the human resources to develop reusable components but also in the form of an organization to manage them. Reusable component designs will in general cost more and will take longer to develop than single purpose designs. The design efforts will also require the talents of some of the best software designers within the company. If the management commitment is not there for this level of effort, a reusable software component program will not succeed.

Initially, savings from reusable software components will need to be reinvested to build up a library of reusable components. An organization will need to be established to maintain control over and to manage these components. The responsibilities of this organization would include:

- Configuration management of reusable software components
- Maintaining documentation sufficient to find and assess the adequacy of the component for various applications
- Monitoring design and development of reusable components
- Helping to identify and select new reusable component candidates
- Tracking problems with reusable components and incorporating user feedback into new designs
- Advocating the use of reusable components wherever possible
- Maintaining reuse statistics on the effectiveness of the reuse program

One last formidable obstacle that must be overcome before reusable software can reach its full potential is programmer acceptance. There is a natural reluctance on the part of most people, including programmers, to accept responsibility for other people's work. This can only be overcome if the programmer has confidence in the capabilities of the reusable component and if he/she is not penalized if what had been a good reusable component does not work in his/her application. The attached list of references shows the high level of interest in reusable software components and some of the wealth of information available in this area.

SUMMARY

The success of reusable software components in flight simulation software development depends upon three major factors. Selection of appropriate

candidates to become reusable software components is the first of these. The successful application of design and development techniques to the real time flight simulation problem domain is the second, and the successful management of the components is the third. Good reusable software components will generally cost more and will take longer to develop than software designed to be used only once. A cost tradeoff analysis needs to be performed to determine the best candidates to become reusable components so that there will be an adequate return on investment.

ABOUT THE AUTHOR

James O'Day is a Senior Member of the Engineering Staff with the Flight Simulation Division of Sperry Systems Management. He is currently the principal investigator on an Ada-based flight simulation IR&D project. Previous responsibilities included team leader on a project simulating an embedded area navigation computer. His military experience includes 1500 hours of flight time as a helicopter pilot in special operations and spacecraft recovery missions. He holds a Masters Degree in Systems Management from the University of Southern California and a BSEE from the U.S. Air Force Academy. Mr. O'Day has also completed the course work requirements towards a Masters Degree in Computer Science at George Washington University.

REFERENCES

1. An Object Oriented Design Handbook for Ada Software, EVB Software Engineering, Inc., Jan 1985.
2. C. Ausnit, et al., "Ada Reusability Guidelines", April 1985, Softech Report, NTIS AD-A161 259.
3. R. Buhr, System Design with Ada, Prentice-Hall, Inc., 1984.
4. G. Booth, "Object-Oriented Development", IEEE Transactions on Software Engineering, Vol. SE-12, No 2, Feb 1986.
5. T. Cheatham, "Reusability Through Program Transformations", IEEE Transactions on Software Engineering, Vol. SE-10, Sept 1984.
6. B. Cox, Object Oriented Programming; An Evolutionary Approach, Addison-Wesley, 1986, ISBN 0-201-10393-1.
7. P. Freeman and A. Wasserman, "Software Engineering Process, Principles, and Goals", Software Design Techniques, fourth edition IEEE Computer Society Press 1983.
8. J. Goguen, "Parameterized Programming", IEEE Transactions on Software Engineering, Vol. SE-10, Sept 1984.
9. J. Goguen, "Reusing and Interconnecting Software Components", Computer, Feb 1986.
10. L. Healy, et al., "Reusable Software - Toward Reconfigurable Trainer Systems", ITEC Conference 1985.
11. E. Horowitz, et al., "An Expansive View of Reusable Software", IEEE Transactions on Software Engineering, Vol. SE-10, Sept 1984.
12. B. Jones, et al., "Issues in Software Reusability", ACM Ada Letters, Vol. IV Num 5.
13. T. Jones, "Reusability in Programming: A Survey of State of the Art", IEEE Transactions on Software Engineering, Vol. SE-10, Sept 1984.
14. R. Lanergan and C. Grasso, "Software Engineering with Reusable Designs and Code", IEEE Transactions on Software Engineering, Vol. SE-10, Sept 1984.
15. S. Litvintchouk and A. Matsumoto, "Design of Ada Systems Yielding Reusable Components: An Approach Using Structured Algebraic Specification", IEEE Transactions on Software Engineering, Vol. SE-10, Sept 1984.
16. M. Mac an Airchinnigh, "Reusable Generic Packages Design Guidelines Based on Structural Isomorphism", Annual National Conference on Ada Technology 1985.
17. G. McKee, "Advanced Tutorial on Designing Ada Packages for Reusability", Tutorial National SIGADA Meeting Nov 1985.
18. Y. Matsumoto, "Some Experiences in Promoting Reusable Software: Presentation in Higher Abstract Levels", IEEE Transactions on Software Engineering, Vol. SE-10, Sept 1984.
19. J. Nissen and Peter Wallis, Portability and Style in Ada, Cambridge University Press, 1984, ISBN 0 521 26482 0.
20. F. Pappas, "Ada Portability Guidelines", Mar 1985, Softech Report, NTIS AD-A160 390.
21. R. Pressman, Software Engineering; A Practitioner's Approach, McGraw-Hill, 1982.
22. T. Standish, "An Essay on Software Reuse", IEEE Transactions on Software Engineering, Vol. SE-10, Sept 1984.
23. R. ST. Dennis, et al., "Measurable Characteristics of Reusable Ada Software", ACM Ada Letters, Vol. VI Num 2, Mar/Apr 1986.
24. D. Whinery and G. Barber, "Analytical Approach to Software Reusability", Annual National Conference on Ada Technology 1985.