CONSIDERING OBJECT ORIENTED TECHNOLOGY IN AVIATION APPLICATIONS

Kelly J. Hayhurst, C. Michael Holloway, NASA Langley Research Center, Hampton, Virginia

Abstract

Few developers of commercial aviation software products are using object-oriented technology (OOT), despite its popularity in some other industries. Safety concerns about using OOT in critical applications, uncertainty about how to comply with regulatory requirements, and basic conservatism within the aviation community have been factors behind this caution.

The Federal Aviation Administration (FAA) and the National Aeronautics and Space Administration (NASA) have sponsored research to investigate and workshops to discuss safety and certification concerns about OOT and to develop recommendations for safe use. Two Object Oriented Technology in Aviation (OOTiA) workshops have been held and numerous issues and comments about the effect of OOT features and languages have been collected. This paper gives a high level overview of the OOTiA project, and discusses selected specific results from the March 2003 workshop. In particular, results in the form of questions to consider before making the decision to use OOT are presented.

Introduction

There is an increasing desire among aviation software developers to use object-oriented technology (OOT), including object oriented modeling, design, programming, and analysis, in the development of aviation applications. These desires are fueled, at least in part, by claims from OOT supporters, such as object orientation "is a more natural form of problem solution and that it results in heavier reuse than its traditional alternatives" [1]. Promises of improved reuse are especially appealing to vendors who build product families for a specialized market, such as aviation, over a long period of time.

Despite claimed cost and quality benefits, few civil aviation applications, especially in airborne systems, have been implemented using OOT. Safety concerns coupled with uncertainty about how to comply with certification requirements have been key obstacles to widespread use of OOT in digital avionics systems.

Compliance with the objectives of RTCA/DO-178B, *Software Considerations in Airborne Systems and Equipment Certification* [2] is the primary means of securing approval of software for use in civil transport aviation products. Similar objectives apply to software used in communications, navigation, and surveillance applications for air traffic management [3]. Neither DO-178B nor DO-278, however, explicitly mentions OOT. Object-oriented (OO) programs that have sought regulatory approval have been required to formulate issue papers to respond to certification concerns.

When DO-178B was published in 1992, structured programming was the predominant technique for organizing and coding computer programs in aviation applications. Although the guidance in DO-178B does not specify a particular development approach, the objectives were formulated largely from the perspective of structured programming. Both developers and certification authorities have raised questions about how the DO-178B objectives are to be satisfied in a project using OOT. Some of these issues are documented in position papers [4, 5] written by the Certification Authorities Software Team (CAST), which helps harmonize software related policy and guidance among international certification authorities.

In an effort to resolve these issues, and to ensure that *all* the important questions are both asked and answered, the Federal Aviation Administration (FAA) enlisted the National Aeronautics and Space Administration (NASA) to help start the Object Oriented Technology in Aviation (OOTiA) project. This project is sponsoring research and conducting workshops designed to develop recommendations for safe use of OOT in compliance with DO-178B.

The OOTiA project was based initially in large part on work conducted by the Aerospace Vehicle Systems Institute (AVSI). AVSI is a research consortium for the aerospace industry working to improve and to reduce the costs of complex subsystems in aircraft. As part of this consortium, Boeing, Honeywell, Goodrich, and Rockwell Collins collaborated on an AVSI project titled *Certification Issues for Embedded Object-Oriented Software*, the goal of which was to mitigate the risk that individual projects face when certifying systems with OO software. The AVSI project proposed a number of guidelines for producing object-oriented software in compliance with DO-178B [6].

In 2001, a committee including representatives from the AVSI project, FAA, and NASA, was formed for the purpose of extending the AVSI work for the benefit of the entire aviation software community. This committee developed the following approach for accomplishing this purpose:

- Set up a web site dedicated to collecting data on safety and certification concerns
- Hold public workshops to which the aviation software community would be invited to discuss concerns
- Document each key concern raised either through the web site or the workshops
- Adapt the AVSI guidelines to address all of the concerns believed by the committee to be valid
- Produce a handbook.

This paper does not attempt to describe fully all of the OOTiA project results to date. Instead, the paper gives only a brief overview of the project, and then discusses in detail the results from one particular session at the March 2003 workshop dedicated to scrutinizing the decision to use OOT.

OOTiA Project Overview

On September 14, 2001, the OOTiA web site http://shemesh.larc.nasa.gov/foot/ was launched by NASA Langley Research Center, and the aviation software community was invited by email[1] to participate in a dialogue about OOT. Individuals could participate by submitting comments, concerns, or issues about OOT to an issue list kept on the OOTiA web site, by attending public workshops organized by the OOTiA committee, and by reviewing products from this effort.

To date, 96 separate concerns[2] about various aspects of OOT have been collected. The web site initially requested that each submittal include a topic, a statement of the concern, and a proposed solution (if known). Neither individual nor company names were recorded with the submittals. No specific guidance was given regarding what could or could not be submitted. Later updates of the web site simply requested that concerns be emailed to a point of contact at NASA Langley.

Each submittal through the web site is added to a list titled "Issues and Comments about Object Oriented Technology in Aviation." This issue list is posted on the web site and updated as new issues are submitted. Every entry that is submitted is added to the list exactly as it is submitted; that is, entries to the list are not edited. Inclusion on the list does not imply that the concern is valid, nor does it imply that the concern is considered important by the OOTiA committee.

Considerable overlap and similarities are evident when reviewing the entries in the issue list. The OOTiA committee originally determined that the following eight topics adequately described most of the issue list: single inheritance, multiple inheritance, reuse and dead/deactivated code, tools, templates, overloading, type conversion, and inlining. Draft papers were written for each of these topics; these papers drew heavily from the original AVSI documents.

In April 2002, a public workshop was held to introduce the OOTiA project, to discuss the draft papers, and to provide an opportunity for people to raise additional concerns about OOT. After this workshop, the individual draft papers were revised and collated into a single document: "Handbook for Object Oriented Technology in Aviation." Also, a ninth topic, traceability, was added, and a paper on the topic included in the draft handbook.

[1] The email distribution list comprised over 900 individuals who have expressed an interest in or attended software related functions sponsored by the FAA.

[2] There are actually 99 entries to the list, but 3 of them are duplicates.

The draft handbook served as the basis for discussion at a second public workshop[3], held in March 2003. Most of the workshop was devoted to individual sessions on specific chapters of the handbook; however, at the request of NASA, a session titled, "Beyond the Handbook," was also held. Whereas the handbook provides guidelines for *how to use* OOT, assuming that the decision to use OOT has already been made, this session provided participants with an opportunity to discuss the questions that should be answered *before* the decision to use OOT is made. The remainder of this paper describes the results from this session.

Beyond the Handbook

At OOTiA Workshop 2, participants in the "Beyond the Handbook" session openly discussed ideas and produced a list of fifty-one questions related to making a decision about whether to use OOT. At the end of the brainstorming session, these questions were reviewed and then grouped under five high-level questions that should be answered before a decision is made. The rest of this section discusses each of these five questions, and the associated issues.

Reality of Benefits

The first question that should be asked and answered is

(1) What are the benefits of OOT compared with current or alternative approaches? And, what evidence exists to support claimed benefits of better, cheaper, faster[4], safer, more reliable, more maintainable, etc.?

OOT has become a popular software development approach within many non-safety-critical industries. OOT is promoted as a technology that allows efficient development of complex systems using reusable modules. Like most new technologies, though, new software technologies often are accompanied by exaggerated claims. OOT is no exception. According to Glass [1], two phenomena typically accompany such claims: "Once the concepts are more thoroughly understood, the benefits turn out to be far more modest than claimed," and "That transition from excessive claims to modest benefits has seldom been accomplished with the aid of evaluative research." That is, practitioners eventually recognize the modesty of the benefits on their own.

Within a group of aviation software engineers, it is not surprising that questions were raised about evidence to support or deny claims. Participants in this session were particularly concerned about finding evidence to support extrapolating the advantages claimed for OOT (even if they *are* real) in non-safety critical systems to safety-critical systems. Because OOT has been around for a relatively long time, one would think there would be an abundance of evidence to promote thorough understanding of OOT benefits. There is an abundance of material, but how much of it qualifies as evidence is debatable.

A quick search of the web for lessons learned and metrics for OOT will net literally thousands of references, from short experience reports to entire books devoted to lessons learned and metrics. A web search for information on empirical studies similarly will yield thousands of references. Studies can be found that support the claimed benefits, such as Basili's results showing reduced defect density and rework with OOT [7], while other studies demonstrate potential problems such as complexity and maintenance problems with the unconstrained use of inheritance [8]. Few empirical studies, however, compare the effectiveness of different software engineering methods, such as comparing OOT to structured programming.

Large-scale empirical studies of software development methods that might provide quantitative assessments are notoriously difficult and expensive. "It costs a lot of money and effort to do controlled experiments, and that is too high a price for most researchers equipped to do such studies, especially in the world of large-scale software" [9]. There are two studies, however, relevant to making the decision about OOT that are worth noting. In a 1994 study [10], Vessey and Conger compared the performance of three different development methods: structured programming,

---

3 Results of the OOTiA workshops are available at http://shemesh.larc.nasa.gov/foot/

4 As an example of these claims, consider a recent Object Management Group (OMG) web cast presentation: "Build IT Better, Cheaper, Faster" available at http://www.omg.org/modeling-webcast.htm. Visited on 28 July 2003.

OO, and Jackson System Development [11]. The results of the study showed that structured methods were easier to apply, at least by novices, than OO methods. A later study by Moynihan showed that functional decomposition, compared with OOT, was easier to understand and enhanced communication between client and developer about requirements [12].

Unfortunately, most empirical studies are open to criticism, both about internal validity (did the experimental treatments really make a difference?) and external validity (to what populations and settings may the results be generalized?) [13]. On the whole, "there is no simple answer regarding the use and performance of OO technologies" [14]. Nonetheless, developers should carefully examine the evidence regarding OOT to better understand potential benefits and risks. If Glass is right, this careful examination should be an adequate substitution for empirical studies.

Project Characteristics

The second important question is

(2) What project characteristics are important with respect to OOT?

Various attributes of a project may help determine whether OOT is an appropriate choice. Some of these attributes are conventional metrics specific to the software product; for example, the size, criticality, and complexity of the software. Other product-specific attributes include the maturity of the software requirements, and the applicability of OOT to the specific problem domain. Concerns were discussed regarding the appropriateness of OOT for all problem domains.

Other attributes of interest relate to the long-term plans for the product. Important considerations here include whether the software is a new product or part of a product family. This would impact upgrade and maintenance requirements. These factors are important when weighing the potential benefits of reuse that OOT may offer.

OOT Specific Resources

Another question that should be asked and answered is

(3) What project resources, specific to OOT, are needed?

Once the project characteristics are known, it is important to evaluate resources specific for implementing OOT. Resources include those relevant to personnel who develop and approve the software product, and those relevant to managing processes and procedures for development and approval.

Personnel resources include OOT-specific training and experience, both at the individual level (such as the software developers and quality assurance personnel) and the corporate level. This includes training and experience with OO methods for modeling, design, analysis and testing, and with OO tools. Note that training and experience is a concern for regulators also, including Designated Engineering Representatives (DERs) within the company and certification authorities responsible for the software approval on the project being reviewed.

Administrative resources include industry standards for OOT, such as those associated with the Object Management Group (OMG) standard for object-oriented modeling with the Unified Modeling Language (UML) [15] and standards for OO source code languages (for example, Ada95, Java, and C++). Other important standards include internal process standards that define life cycle activities and data associated with OOT and how those map to activities and data specified in DO-178B.

OO tools are another important resource to consider. Some OO tools introduce new levels of abstraction, such as the visual model level, that may not directly correspond to abstraction levels (high- or low-level requirements or design) in DO-178B. Factors to consider here include compatibility of new OO tools with existing tools, notations, and processes; configuration management; and qualification costs.

The project characteristics together with the OOT specific resources within a company will influence the level of involvement, or degree of oversight, that the FAA has with a project. This is a non-trivial consideration with respect to both time and cost. The level of FAA involvement will dictate the number of software reviews, the stages

of involvement, and the nature of the review [16]. This level of regulatory involvement is closely related to the fourth of the high-level questions raised at the workshop.

Regulatory Guidance

The fourth question is

(4) How should regulatory guidance, including DO-178B and the OOTiA handbook, be applied in a practical project?

This question is really an abstraction of two more specific questions:

- Are all of the objectives in DO-178B compatible with OOT?
- How should the handbook be applied to a practical project, and is the handbook adequate?

As mentioned previously, the FAA is sponsoring the development of the OOTiA handbook to provide information specific to meeting the DO-178B objectives when using OOT. Some participants in the brainstorming session argued that the existing guidance in DO-178B is sufficient to accommodate approval of an OO program. Some questioned the wisdom of generating an OOT-specific handbook, and wondered whether that implied the need for additional method-specific handbooks. Other participants, including some regulators, however, argued in favor of the benefits that additional clarification and guidelines might provide in the short term.

The handbook is not intended to be official FAA policy or guidance [17], but the handbook will almost certainly influence the approval process for an OO program. The handbook does not eliminate the need for compliance with DO-178B, but instead provides guidelines for how to use OOT to comply with the DO-178B objectives. A significant portion of the handbook is devoted to patterns intended to ensure this compliance.

If the handbook is to be used effectively by developers and regulators, then it must provide clear guidelines. Clear communication of regulatory requirements, among regulators and between regulators and software developers, has been a perpetual problem for aviation software development [18]. Regulators and software developers must both understand the requirements the system must satisfy for it to be approved, and how the system will be shown to satisfy these requirements [19]. Misunderstandings can result in substantial cost and schedule problems.

Technical Challenges

The final, and perhaps most difficult, question that should be asked and answered by anyone considering using OOT is

(5) What are the technical challenges in applying OOT to ensure the appropriate level of integrity required for the project?

Specific questions raised in the session concerned how well the essential elements of software engineering can be done using OOT to ensure the appropriate level of integrity. Most of the questions within this grouping were about requirements, verification, or safety.

Requirements

Several questions asked whether OOT is the correct approach for requirements development and implementation. In particular, questions were raised about the effectiveness and appropriateness of the OO approach to requirements development, which is based on use cases. The discussion involved the difference between the functional decomposition (or structured programming) approach and object-orientation.

With functional decomposition, the typical programming unit is some form of subprogram, such as a function, subroutine, or procedure. Each subprogram typically performs a single specific function, where good programming practice calls for maximizing functional cohesion within a subprogram and minimizing coupling between subprograms. Applications are built by sequencing these functional building blocks—"first do this, then do that." Verification, in turn, starts with the functionality of an individual subprogram and works its way up by testing increasing levels of functionality.

In contrast to functional decomposition, OOT focuses on objects and the operations performed by or to those objects. In an OO program, a class,

which is a set of objects that share a common structure and a common behavior, is the structural element most comparable to a subprogram. Operations related to a given functional requirement often are distributed among objects associated with different classes.

The fundamental goal of the approval process, as guided by the Federal Aviation Regulations, is to provide assurance of the intended functionality and provide assurance that there is no unintended functionality. DO-178B does not refer specifically to subprograms or functional units, but it does organize guidance for development and verification around the decomposition of requirements from high-level requirements to low-level requirements to source code. This seems reasonable since the system level requirements, which are the source of the high-level software requirements, are written by and large from a functional perspective. Many of the development and verification objectives in Annex A of DO-178B are specific to high or low level requirements and code.

Typically, requirements for OO systems are developed with use cases, scenarios, and various diagrams such as class, object, and activity diagrams. Determining how to map these modeling components, and their subsequent refinements, onto the DO-178B objectives was thought by session participants to be difficult. Some participants questioned whether such an approach is even appropriate for safety-critical applications. Requirements definition by any method is a significant challenge to developing a correct and safe system [20]. Developers should consider whether OOT makes this challenge more difficult.

Verification

In addition to the questions raised about the suitability of OOT for requirements development, a similar number of questions were raised about verification. The questions about verification are not unrelated to the concerns raised about requirements. According to Alexander, "object oriented programs are generally more complex than their procedural counterparts. This added complexity results from inheritance, polymorphism, and the complex data interactions tied to their use. Although these features provide power and flexibility, they increase complexity and require more testing" [21]. Many in the brainstorming session echoed this sentiment.

Several of the questions discussed in the session sought to explore the extent that OO software can be verified:

Can we analyze OO software?

Can we adequately test OO software?

Can we determine the error cases unique to OOT?

Other questions focused on more specific aspects of verification, especially analysis issues such as source to object code traceability, and control and data flow analysis. Several participants in the session argued for the application of static analysis and formal methods.

With respect to verification, many participants acknowledged the need for additional research to better understand error classes that are unique to OOT, such as research by Offutt [22], and to better understand the extent that existing methods are adequate for verifying OOT.

Safety

The final technical challenge mentioned in the questions concerns the ability to conduct system and software safety assessment. Participants discussed whether system and safety assessments can be easily and accurately derived from an OO program. Current safety analysis is often based on determining that a function, as implemented, is both correct and safe. OOT complicates this analysis because the operations related to a function can be widely distributed throughout the objects, making the function difficult to trace.

Workshop participants are not alone in questioning safety analysis on OO systems. In a related discussion on the safety-critical mailing list, participants discussed the importance of a functional perspective to safety analysis. In the mailing list discussion, Nancy Leveson argued that engineers find that functional decomposition is a more natural approach to the design of control systems, and "That naturalness translates into easier to understand and review, easier to design without errors, easier to analyze to determine whether the system does what the engineer want and does it safely" [23]. Others suggested design approaches

in OOT to enhance the ability to do the safety analysis.

Even though safety analysis is not part of the life cycle activities specified in DO-178B, the effect of OO design and implementation on safety analysis should be carefully considered.

As noted in [4], "Developers should carefully weigh their program needs with the benefits and risks of OOT." This includes examining the evidence relevant to the benefits of OOT, project characteristics and resources, and the technical challenges.

Summary

Object-oriented technology is immensely popular within certain segments of the software community, but popularity does not guarantee propriety, especially for safety-critical systems. An intelligent decision about whether to use OOT must be based on answering specific questions about OOT's propriety for particular applications and levels of integrity. This paper has presented some of the questions proposed by members of the aviation software community as important to ask before using OOT. Only if each of these questions is answered satisfactorily should an aviation software developer commit to using OOT. If this decision is made, then the OOTiA handbook, once it is completed, will provide guidelines to assist developers in obtaining approval from certification authorities for OOT software.

Acknowledgement

The questions and concerns raised in this paper were taken from the results of the "Beyond the Handbook" brainstorming session held at the OOTiA Workshop 2, in March 2003. We thank the session participants for their candor, insight, and diligence in raising and discussing these questions.

References

[1] Glass, Robert L, May/June 2002, "The Naturalness of Object Orientation: Beating a Dead Horse?" *IEEE Software*, pp. 103-104.

[2] RTCA, Inc., December 1992, Software Considerations in Airborne Systems and Equipment Certification, RTCA/DO-178B, Washington, D. C.

[3] RTCA, Inc., 5 March 2002, Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance, RTCA/DO-278, Washington, D. C.

[4] Certification Authorities Software Team (CAST), January 2000, Object-Oriented Technology (OOT) in Civil Aviation Projects: Certification Concerns, Position Paper CAST-4, available at http://av-info.faa.gov/software/CAST/cast-4.rtf. Visited on 29 July 2003.

[5] Certification Authorities Software Team (CAST), January 2002, Use of the C++ Programming Language, Position Paper CAST-8, available at http://av-info.faa.gov/software/CAST/cast-8.rtf. Visited on 29 July 2003.

[6] Aerospace Vehicle Systems Institute, 31 October 2001, Guide to the Certification of Systems with Embedded Object-Oriented Software, version 1.2

[7] Basili, V., L. Briand and W. Melo, 1996, "How Reuse Influences Productivity in Object-Oriented Systems," *Communications of the ACM*, vol. 39, no. 10, pp. 104-116.

[8] Wood, M, J. Daly, J. Miller, and M. Roper, 1999, "Multi-Method Research: An Empirical Investigation of Object-Oriented Technology," *The Journal of Systems and Software*, no. 34, pp. 13-26.

[9] Peterson, Ivars, 1995, *Fatal Defect: Chasing Killer Computer Bugs.* Random House, Inc., New York

[10] Vessey, Iris, and Sue A. Conger, May 1994, "Requirements Specification: Learning Object, Process, and Data Methodologies," *Communications of the ACM*, vol. 37, no. 5, pp. 102-113.

[11] Jackson, M., 1983, *System Development*, Prentice-Hall, Englewood Cliffs, New Jersey.

[12] Moynihan, Tony, 1996, "An Experimental Comparison of Object-Orientation and Functional-Decomposition as Paradigms for Communicating System Functionality to Users," *The Journal of Systems and Software*, vol. 33, pp. 163-169.

[13] Campbell, Donald T., Julian C. Stanley, 1963, *Experimental and Quasi-experimental Designs for Research*, Houghton Mifflin Company, Hopewell, New Jersey.

[14] Briand, L., E. Arisholm, S. Counsell, F. Houdek, and P. Thévenod-Fosse, 1999, "Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions," Technical Report ISERN-99-12.

[15] Object Management Group, March 2003, OMG Unified Modeling Language Specification, Version 1.5, formal/03-03-01.

[16] FAA Aircraft Certification Service, June 1998, Conducting Software Reviews Prior to Certification, Job Aid, available at http://av-info.faa.gov/software/Job_Aids/jobaid.rtf. Visited on 29 July 2003.

[17] Rierson, Leanna K., 27 March 2003, FAA's Next Steps for OOTiA, presented at the Object Oriented Technology in Aviation Workshop 2, available at http://shemesh.larc.nasa.gov/foot/next-steps-end.ppt. Visited on 29 July 2003.

[18] Hayhurst, Kelly J; Cheryl Dorsey; John Knight, Nancy Leveson, G. Frank McCormick, August 1999, Streamlining Software Aspects of Certification: Report on the SSAC Survey, NASA/TM-1999-209519.

[19] Hayhurst, Kelly J., C. Michael Holloway, 27-29 November 2001, "Challenges in Software Aspects of Aviation Systems," *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, Greenbelt, MD, pp. 7-13.

[20] Hanks, Kimberly S., John C. Knight, Elisabeth A. Strunk, 27-29 November 2001, "Erroneous Requirements: A Linguistic Basis for Their Occurrence and an Approach to Their Reduction," *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, Greenbelt, MD, pp. 115-119.

[21] Alexander, Roger T., September/ October 2001, "Improving the Quality of Object-Oriented Programs," *IEEE Software*, pp. 90-91.

[22] Offutt, Jeff, Roger Alexander, Ye Wu, Quansheng Xiao, Chuck Hutchinson, November 2001, "A Fault Model for Subtype Inheritance and Polymorphism," *The 12th IEEE International Symposium on Software Reliability Engineering*, Hong Kong, PRC, pp. 84–95.

[23] Leveson, Nancy, 2002, "Re: object-orientation vs. safety-critical" in Safety-Critical Mailing List, archived at http://www.cs.york.ac.uk/hise/safety-critical-archive/2002/0203.html. Visited on 28 July 2003.